

Soloist

מסמך עיצוב ותיכון

גרסה 1.0



חנן וולט
09/10/2020
האוניברסיטה הפתוחה

תוכן עניינים

4	1	כללי
4	1.1	עקרונות הפיתוח
5	2	הנחות עבודה
6	3	מוסכמות שמות במערכת
6	3.1	מוסכמות שמות הטבלאות בסיס הנתונים
6	3.2	מוסכמות שמות רכבי הפיתוח
8	4	ארכיטקטורת המערכת
8	4.1	סקירה כללית
9	4.2	שכבות המערכת
10	5	שכבה הלוגיקה העסקית (BLL)
11	5.1	יצוג ישוות מוסיקליות (Music Theory)
12	5.1.1	משך שהייה (Duration)
15	5.1.2	תווים (Notes)
20	5.1.3	אקורדים (Chords)
22	5.1.4	תיבות (Bars)
24	5.1.5	מרווחים (Intervals)
25	5.1.6	מחלקה מפעל לישויות מוסיקליות (MusicTheoryFactory)
26	5.1.7	הרחבות ושירותים נוספים
28	5.2	תת-שכבה שירותים MIDI (MIDI)
28	5.2.1	רकע – פרוטוקול MIDI וקובצי SMF
29	5.2.2	ארכיטקטורת תת-שכבה שירותים MIDI
30	5.2.3	אוסף מספרי רצועות (MelodyTrackIndex Enumeration)
31	5.2.4	אוסף כלי נגינה (MusicalInstrument Enumeration)
31	5.2.5	מנשך רצועת MIDI (IMidiTrack Interface)
32	5.2.6	מנשך קובץ MIDI (IMidiFile Interface)
34	5.2.7	מחלקה מעטפת לקובץ MIDI (DryWetMidiAdapter Class)
35	5.2.8	מחלקה מעטפת לרצועת MIDI (DryWetMidiTrackAdapter Class)
36	5.2.9	מחלקה מפעל לישויות MIDI (MidiFactory Class)
37	5.3	הלחנת מנגינות (Composition Service)
38	5.3.1	מחלקה קונטסט הלחנה (CompositionContext Class)
43	5.3.2	מחלקהמלחין אבסטרקט (Composer Abstract Class)
47	5.3.3	اللحנה عم الگוריתם גנטי
61	5.3.4	اللحנה عم الگורייתמים נוספים
62	5.3.5	מפעלייצירתמלחינים
63	5.3.6	אינטראקטיבית בין היישוות בתהליך הלחנה
65	5.3.7	הרחבות (ComposerExtensions)

65	אוסף קבועים (Enums)	5.3.8
68	שכבת הגישה לנוטונים (DAL)	6
70	בסיס הנתונים (Database)	6.1
70	דיאגרמת טבלאות	6.1.1
70	פירוט הטבלאות	6.1.2
72	מודלים (Domain Models)	6.2
72	מודל משתמש (ApplicationUser Model)	6.2.1
73	מודל שיר (Song Domain Model)	6.2.2
74	מאגרים (Repositories)	6.3
75	מנשך מאגר (IRepository Interface)	6.3.1
76	מנשך מאגר שירים (ISongRepository Interface)	6.3.2
77	מחלקה-אב גנריית למימוש מנשך המאגר הגנרי (EFRepository Class)	6.3.3
78	מחלקה מאגר שירים (SongRepository Class)	6.3.4
80	ניהול יחידות עבודה (Unit Of Work)	6.4
81	מנשך ייחידת עבודה (IUnitOfWork Interface)	6.4.1
82	מחלקה ייחידת עבודה (EFUnitOfWork Class)	6.4.2
83	מייפוי בין אובייקטים לרשומות (EntityFramework ORM)	6.5
84	מחלקה קונטסט מסד הנתונים (ApplicationDbContext Class)	6.5.1
85	מחלקה מייפוי ישות שיר לatable (SongConfiguration Class)	6.5.2
86	מיגרציות (Migrations)	6.6
86	שכבת התצוגה (PL)	7
87	צרכים תוכן האפליקטיבי על השרת (App_Data)	7.1
87	לוגים (Logs)	7.1.1
87	שירים (Songs)	7.1.2
88	מנגינות (Outputs)	7.1.3
88	סמינר (Seminar)	7.1.4
88	קונפיגורציות אתחול אפליקציה (App_Start)	7.2
88	הגדרת חבילות (BundleConfig)	7.2.1
88	רישום מחלקות פילטר (FilterConfig)	7.2.2
89	הגדרת שירות זיהוי (IdentityConfig)	7.2.3
89	הגדרת שירות הרשאה (Startup.Auth)	7.2.4
90	בקרים (Controllers)	7.3
91	בקר דף הבית (HomeController)	7.3.1
92	בקר שירים (SongsController)	7.3.2
94	בקר הלחנה (CompositionController)	7.3.3
96	בקר ניהול (AdminController)	7.3.4

<p>96 בקרי ניהול חשבונות משתמשים</p> <p>96 פילטרים (Filters) (Filters) 7.4</p> <p>98 פילטר טיפול בחיריגות (CustomExceptionHandler) 7.4.1</p> <p>99 פילטר טיפול בבקשת (ActionFilters) 7.4.2</p> <p>100 מחלקות ולידציה (Validations) 7.5</p> <p>100 בדיקת תקינות קבצים (FileUploadValidation) 7.5.1</p> <p>102 בדיקת תקינות טווח צלילים (PitchRangeValidation) 7.5.2</p> <p>103 מודלים לטפסים (ViewModels) 7.6</p> <p>103 מודל-תצוגה עבור שיר (SongViewModel Class) 7.6.1</p> <p>104 מודל-תצוגה עבור הלחנת מנגינה (CompositionModel Class) 7.6.2</p> <p>105 מודל-תצוגה עבור ניהול חשבונות משתמשים</p> <p>106 מסכים וטפסים (Views) 7.7</p> <p>107 אינטראקציה בין רכיבי ה-MVC 7.8</p>	<p>7.3.5</p> <p>7.4</p> <p>7.4.1</p> <p>7.4.2</p> <p>7.5</p> <p>7.5.1</p> <p>7.5.2</p> <p>7.6</p> <p>7.6.1</p> <p>7.6.2</p> <p>7.6.3</p> <p>7.7</p> <p>7.8</p>
108 דפוסי עיצוב.....	
<p>108 דפוסי ייצור (Creational Patterns) 8.1</p> <p>108 דפוס מפעל (Factory Pattern) 8.1.1</p> <p>110 דפוסים מבניים (Structural Patterns) 8.2</p> <p>110 דפוס מעטפת מתאמת (Adapter Pattern) 8.2.1</p> <p>111 דפוסים התנהגותיים (Behavioral Patterns) 8.3</p> <p>111 דפוס אסטרטגיה אלגוריתמית (Strategy Pattern) 8.3.1</p> <p>112 דפוס תבנית אלגוריתמית (TemplateMethod Pattern) 8.3.2</p> <p>113 תבניות ודפוסים ארכיטקטוניים 8.4</p> <p>113 מודל-תצוגה-בקраה (MVC) 8.4.1</p> <p>113 מיפוי אובייקטיבי-רלציוני (ORM) 8.4.2</p> <p>114 (Repository & Unit Of Work) DB 8.4.3</p>	<p>8</p> <p>8.1</p> <p>8.1.1</p> <p>8.2</p> <p>8.2.1</p> <p>8.3</p> <p>8.3.1</p> <p>8.3.2</p> <p>8.4</p> <p>8.4.1</p> <p>8.4.2</p> <p>8.4.3</p>
115 תיאור שינויים עתידיים	
<p>115 שינויים תשתיתיים 9.1</p> <p>115 שינויים תשתיתיים הקשורים ל-DB 9.1.1</p> <p>116 שינויים בפלטפורמה 9.1.2</p> <p>116 שינויים אפליקטיביים/פונקציונאליים 9.2</p> <p>116 הוספה אלגוריתם הלחנה 2.1.9</p> <p>117 עדכון האלגוריתם הגנטי הקויים 9.2.2</p>	<p>9</p> <p>9.1</p> <p>9.1.1</p> <p>9.1.2</p> <p>9.2</p> <p>2.1.9</p> <p>9.2.2</p>
118 נספח: רקע תאורטי של המוסיקה המערבית	

1 כללי

מסמך זה מכיל את פרטי מימוש העיצוב והຕיכון של מערכת Soloist כפי שאופינה במסמך האפיון והניתוח –

<https://github.com/cwelt/Soloist/blob/master/Design/Documents/pdf/OOA-Object-Oriented-Analysis.pdf>

טכנולוגיות מרכזיות

מערכת זו ממומשת בשפת C# על פלטפורמת ה-.NET Framework Web באמצעות ASP.NET MVC 5. כAPPLICATION.NET Web במאצעות CSS, HTML ו-Javascript עבור ניהול צד השירות, ו-Bootstrap עבור ניהול צד הלקוח, ללא שימוש בספריות frontend ייודיות, למעט השלמה של פונקציונאלויות בסיסית עם JQuery.

עבור ניהול הגישה למסדי נתונים רלציוניים (Relational databases) והמייפוי בין הרשומות במסדי נתונים אלו לבין היחסיות העסקיות המתאימות להן בעולם התכנות המונחה עצמים, נעשה שימוש ב-6 Entity Framework – רכיב ה-ORM הstdnetדרטי מבית Microsoft.

כמו כן, השימוש כולל שימוש גם בספריות חיצונית המספקות שירותים חיוניים מסוימים כמו Autofac, DryWetMIDI, MIDI, ספריית(arouyi), ספריית SendGrid עבור שליחת מיילים. ניהול תלוויות (Dependency Injection & IoC).

1.1 עקרונות הפיתוח

להלן העקרונות המנחים שנלקחו בחשבון בהחלטות על אופן עיצוב ומימוש המערכת, והאסטרטגיות המרכזיות שננקטו בכך לישם אותם –

1. **משק נח למשתמש** – המערכת צריכה להיות קלה ונוחה לתפעול גם למשתמשים שאינם מכירים כלל את המוסיקה המערבית, ובפרט אינם מכירים תווים, סולמות וכו'. לשם כך המערכת צריכה לעתוף ולהסתיר את כל המידע התאוריתי המוסיקלי וכל הלוגיקה האלגוריתמית של הקומפוזיציה מאחרי משק מופשט – "קופסה שחורה", כך שימושים יכולים לחבר מנגינות חדשות על-פי טעums בכמה לחיצות כפתור.

2. **גמישות לשינויים** – ארכיטקטורת המערכת צריכה לאפשר שינויים ושיפורים עתידיים ותחזקה שוטפת בנסיבות יחסית ללא עדכונים מהותיים. לטובות עיקרונו זה, בכל חלקי הפיתוח מוטמעים דפוסי עיצוב (Design Patterns) ועקרונות ה-SOLID עבור תכונות מונחה עצמים.

3. **אי-תלות בפלטפורמה** – המערכת צריכה להיות חופשית מתלות ברכיבי החומרה, מערכת הפעלה וסוג היישום (אפליקציית שולחני / Web / אפליקציה (Smartphone)). לטובות אי-תלות זו המערכת עוצבה בחלוקת לשכבות – שכבת הצוגה/יישום, שכבת הלוגיקה העסקית, ושכבת ניהול הגישה לנಟונים. בשלב ראשוני הוחלט לפתח שכבת הצוגה כישום Web, כך שייתאפשר שימוש במערכת מכל מחשב נייח/נייד/סמרטפוןים ללא תלות במערכת ההפעלה וברכיב החומרה. אם בעתיד יידרש משק משתמש ייודי חדש/נפרד למחשב שולחני / סמרטפון / טאבלט ועוד, ניתן יהיה לעשות שימוש בתשתיות הקיימת של שכבות ניהול הגישה לנוטונים והלוגיקה העסקית וرك לספק מימוש חדש לשכבת הצוגה/יישום.

2 הנחות עבודה

הפיתוח נעשה תחת הנחות העבודה הבאות –

חווארה

- המערכת מיועדת להפעלה בעיקר ממחשבים PC. אמנים היישום הוא Web-י, כך שיש תמיכה בכל המכשירים ללא תלות במערכת ההפעלה ובסוג המכשיר (PC, Smartphone, Tablet,...).
- אולם הפיתוח מוכoon למחשבים PC, אז קיבלת תצוגה מיניבית מומלץ לעשות שימוש במערכת מתוך מחשבים נייחים/ניידים ולא מתוך טלפונים חכמים.

הנחות על קובצי MIDI

הפורמט המרכזי לקובצי השירים הוא קובצי MIDI. דיקוק מנווק לבחירה בפורמט זה מפורט במסמך האפיון והנחיות ובמסמך זה בסעיף רקע – פרוטוקול MIDI וקובצי SMF. להן הנחות נוספות בדבר תוכן הקבצים –

- קבצי MIDI בפורמט 0, המרכזים את תפקידי כל כלי הנגינה ביחד עם Metadata של נתוני כוורתה הקובץ ברצואה יחידה, אינם נתונים ע"י המערכת. המערכת תומכת רק בפורמטים 1 ו-2, המבצעים הפרדה בין הרצאות השונות וכן בין רצאות הנגינה לבין רצעת הכותרת. פירוט נוספת על פורמטים בקובץ MIDI ניתן בקישור הבא –

http://www.music.mcgill.ca/~ich/classes/mumt306/StandardMIDIfileformat.html#BM2_2

- המערכת תומכת ב- 16 רצאות לכל היוטר (לא כולל את רצעת הכותרת). במידה ויש צורך לטפל בקובץ MIDI הcoilל בנוסף לרצאות הכותרת 17 רצאות או יותר, יש לעדכן תחילת את הקובץ כך שירכז את אירורי-h-MIDI ב-16 רצאות נגינה לכל היוטר.

- עבור כל הנגינה שיוגדר לנגן מגניות חדשות שייווצרו ע"י המערכת, יש תמיכה בכל נגינה מסוט הערכיים של General Midi בטוווח 0 עד 112 (עד משפחת הכלים האתניים כולל). טווח זה כולל את כל הנגינה הסטנדרטיים, ללא אפקטים קוליים סינטטיים ולא כל הקשה. פירוט נוספת על ערכי-h-MIDI ניתן בקישור הבא –

http://www.music.mcgill.ca/~ich/classes/mumt306/StandardMIDIfileformat.html#BMA1_4

הנחות על מגבלות ייצוג / זיכרון

המערכת יודעת מנקודות הנחה שהשירים המועלים למערכת הם בעלי הגדרות קבועות ומוגדרות "הגינויים", במובן שהם ניתנים לנגינה ע"י בן אדם. תחת הנחות סבירות אלו, מוגדרות המוגבלות הבאות –

- ערך ה-BPM – Beats Per Minute – שמנדריך את מספר הפעימות בדקה של השיר, וכונזרת מכון קובע את מהירות הקצב, מיוצג ע"י byte, משמע שטוווח הערכיים שלו נע בין 0 ל-255.
- משכי שהייה (כגון רביע, שמינית וכו') המיוצגים במערכת ע"י שני משתנים נפרדים המרכיבים את המונה והמכנה, מיוצג גם הוא במונה ובמכנה ע"י משתני byte וטוווח ערכיים אלו בהתאם.

3 מוסכמות שמות במערכת

סעיף זה מתאר את מוסכמוות השמות שניתנו לרכיבי הפיתוח השונים במערכת (משתנים, מחלקות, מנשקים, פונקציות ועוד), וכן מוסכמוות השמות לטבלאות בסיסי הנתונים.

3.1 מוסכמוות שמות הטבלאות בסיס הנתונים

שמות הטבלאות והשדות יהיו שתיים ב-Pascal Case (אות רישית גדולה בתחילת כל מילה). שמות טבלאות יהיו ברבים וייהו בעלי שמות תואמים עד כמה שניתנו למחלקות התואמות להן ברכיבי הפיתוח, במידה וישן כאלה. למשל עבור מחלוקת Song המיצגת שיר, שם הטבלה יהיה Song.

3.2 מוסכמוות שמות רכיבי הפיתוח

באופן כללי, לאחר שהמערכת ממומשת בשפת C#, בחירת השמות נעשתה בהתאם לקווי הנקה הסטנדרטיים של Microsoft עבור פיתוחים מעל ה-.NET Framework. ב-C#.NET אומץ – עקרונות המפורטים בקישורים הבאים –

- i. [C# Coding Standards and Naming Conventions](#)
- ii. <https://docs.microsoft.com/en-us/dotnet/standard/design-guidelines/naming-guidelines>

בסעיף 'מוסכמוות כתיבה' שלහלן נביא את עיקרי הדברים והבחירות שבוצעו בפועל בפיתוח המערכת מבין האפשרויות המומלצות בקווי הנקה הסטנדרטיים, כפי שמתוירים בקישורים שלעיל.

הערות מקדיימות

1. Beispiel: בתיאור מוסכמוות הכתיבה נשתמש בטרמינולוגיה Pascal Case, ו- Camel Case. Pascal Case היא צורת כתיבה שבה אות רישית (אות גדולה) פותחת כל שם עצם/פועל/מילה בשם (למשל: ReadChordsFromFile, Camel Case). היא צורת כתיבה שבה שם העצם/פועל/מילה הראשונים כתובים דוקא עם אות קטנה, ואילו המילים העוקבות פותחות עם אות רישית (אות גדולה). (למשל: chordProgressionFilePath). במקרה שתהיה זה ברור מידית בהקשר הרלוונטי, בהמשך פשוט נרשות את המושגים שלעיל בזורת הכתיבה שהם מייצגים כך: camelCase ו-PascalCase.

2. Beispiel: התיאור ברשימת מוסכמוות הכתיבה המפורטוות בהמשך הינו תיאור של מוסכמוות, ולא הנחיות מחיבות – אין כאן נכון ולא נכון, אלא פשוט קוווי הנקה שצדאי להיצמד אליהם כדי שהקוד יהיה אחיד וברור ישרות מתוך המוסכמוות. ואכן, ישנו מקרים חריגים יוצאים מן הכלל שבהם לא מיושמים המוסכמוות שלעיל. למשל –

❖ **משתנים פרטיים קבועים (private const...)** – שם גם פרטיים וגם קבועים, אז מצד אחד ע"פ המוסכמוות הם אמורים להירשם בפורמט משתנים פרטיים עם קו-תחתיו ו- _somePrivateFieldNamecamelCase, ומצד שני הם קבועים ולכן אמורים להירשם ללא קו-תחתיו תחيلي וב- PascalCase. במקרים סטוריים שכאלה, אין כלל קבוע, לעיתים השדה יירשם במוסכמה הראשונה (אם למשל נראה חשוב להציג שמדובר בשדה פרטי שאינו אמר או היה להיות כלל לרכיבי פיתוח חיצוניים) ולעתים השניה (אם מדובר שעובד בקבוע חשוב יותר להדגשה מהעובד שמדובר במשתנה שהוא פרטי לגוף המחלוקת).

❖ **בפלטפורמת ASP.NET MVC לפיתוח יישומי ה-Web**, נהוגה דוקא מוסכמת כתיבה שונה עבור משתנים פרטיים שמקבלים שם דוקא שמות עם camelCase ללא קו-תחתיו תחيلي, למשל: private DbContext db (ולא _db private DbContext), על-כן במרחב השמות של פיתוח הממשק ה-Web-י למשל, יתכוון חריגות קלות שכאלה ממוסכמוות הכתיבה שלעיל.

מעבר לקרים חריגים שיוצאים מון-הכללפה ושם כמו שתי הדוגמאות שלעיל, קור-המקור של המערכת נצמד באופן עקבי למוסכמוות הכתיבה המפורטוות מיד בעמוד הבא.

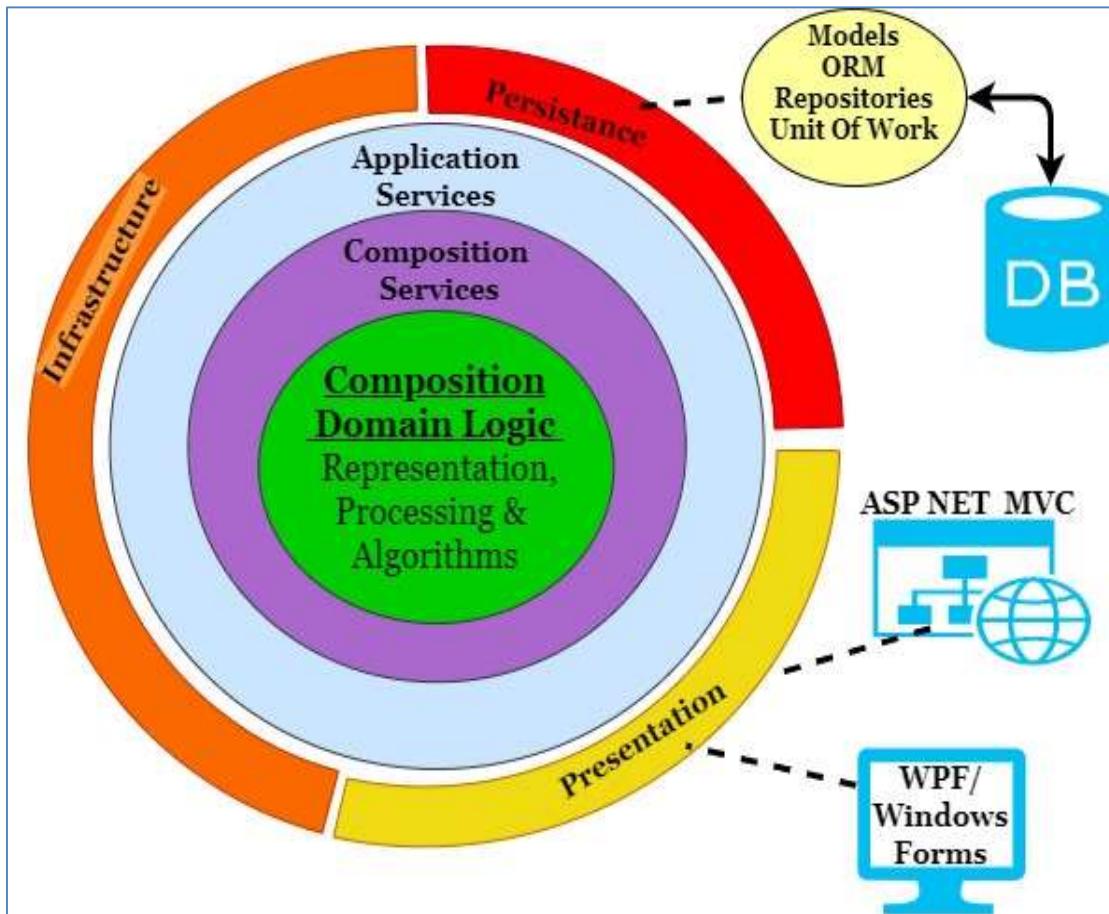
רשימת מוסכמתו הכתיבה של שמות רכיבי הפיתוח –

- **מרחבי שמות (Namespaces)** : CW.Ubero<ProjectName> יתחלו עם הקידומת Soloist¹, Ubero שם המחבר¹, Soloist שם המערכת, ולאחר מכן שםuproject (שם יחידת h-Assembly) המרכזי של רכיבי הפיתוח השיכים למורחב השמות זהה. למשל Ubero שכבת הגישה לנתונים המרוכזות בפרויקט DataAccess, כלל מרחבי השמות בפרויקט יהיו תחת עץ המשפחה של מרחב השמות : CW.Soloist.DataAccess. מוסכם זו חלק גם על ההיררכיה הפנימית, למשל, הלוגיקה לניהול Repositories בשכבות הנתונים יכולה להיות מרוכזות תחת מרחב השמות DataAccess.CW.Soloist.DataAccess.Repositories. כך ברור מידית מהקשר מהו מקור הספרייה/רכיב פיתוח (ספרייה חיצונית/ספרייה השיכת למערכת) וכן במקורה שאכן מרחב השמות שייך למערכת, ניתן לגוזר מיד מהשם מהי שכבה הרלוונטית, ולאיזה ענף בהיררכיה הפנימית של אותה שכבה רכיב הפיתוח שייך.
- **רכיבי פיתוח עם שמות בצורה PascalCase** : מרחבי שמות (Namespaces), מחלקות, מנשכים (Interfaces), מethods (שיטות), בנאים (Properties), Constructors (Konstruktoren), משתני הפניה (Properties), משתנים מאפיינים (Constructors), משתנים קבועים וכן טיפוסי קבועות בנות-מנייה של קבועים (Delegates) – כל אלו נכתבם בשיטת PascalCase (Enumerations).
- **רכיבי פיתוח עם שמות בצורה camelCase** : פרמטרים למethods, משתנים מקומיים, ושדות (משתני מופע שאינם נמנים בין Properties של המחלקה/מנשך) – כל אלו נכתבם בשיטת camelCase.
- **שמות שימושיים** : להוציא אינדקסים שככל תפקדים הוא למנות איטרציות בלולאות, כל יתר רכיבי הפיתוח – משתנים, שדות, מethods, וכוי ישאו שם שימושי המבהיר היבט את טיב הרכיב.
- **משתנים פרטיים ייכתו עוקו תחתון תחילי '_** (underscore character) לפני שם המשתנה private Compositor _compositor ; למשל Ubero משתנה פרטי composito;
- **מנשכים (Interfaces)** ייכתו עם תחילית של האות הרישית I בצד להציג שמדובר ב-interface. לדוגמה : אם נרצה להגדיר פונקציונליות של הלחנה (קומפוזיציה) תחת ישות Compositor שתהייה אחראית על פונקציונליות זו, ונבחר לרכז את ההגדרה המופשטת של פונקציונליות זו במנשך ייעודי, שם הולם למנשך שכזה יכול להיות למשל ICompositor.
- **מетодות יקבלו שמות שימושיים שמציינים את פעולה בגוף שני**. שם הולם למשל, למודה שאחראית על הלחנה הוא Compose. שמות טובים יבהירו מידית את מהות אקט המודה. למשל המודה ChangePitchForARandomNote
- **משתנים בוליאניים ומethodות בוליאניות** ייכתו עם שם המיציג את מהות הפסוק הלוגי / אמרה לוגית שערק האמת שלהם הוא זה שמוחזר עיי' המתודה הבוליאנית / מתוחזק במשתנה הבוליאני. למשל המתודה הבוליאנית IsPitchRangeValid אחראית לבדוק האם טווח מנעד הצלילים תקין או לא. אם כן, המתודה תחזיר ערך אמת True, ואילו אם הטווח אינו תקין המתודה תחזיר ערך אמת False. פונקציונליות זו משתמשת ישירות מהמוסכמה.
- **רכיבי פיתוח מרכזיים המשתתפים בימוש דפוסי עיצוב** ייכלו בשםם את תפקודם בكونטיקט שימוש דפוס העיצוב. למשל – מחלקות שאחראיות על ייצור אובייקטים אחרים ייכלו בשםם את המושג Factory, ומחלקה האחראית לעטוף איזשהו מנשך, להסתירו ולספק במקומו מנשך אחר, ייכלו בשםם את המושג Adapter.

4 ארQUITקטורת המערכת

4.1 סקירה כללית

לטובת הסדר הטוב במערכת, המודולריות שלה, אמינותה והגמישות שלה לתחזוקה שוטפת ושדרוגים עתידיים, המערכת נבנתה בחלוקת לשכבות לוגיות ופיזיות שונות בהשראת ארQUITקטורת ה-3-Tier Architecture. אולם להבדיל מהגישה הקלאסית של ארQUITקטורת שלוש השכבות – שכבת תצוגה, שכבת אפליקציה ושכבת גישה לנוטונים שבה ב"ד"כ שכבת הגישה לנוטונים היא שכבה המרכזית שעיצוב המערכת נעשה סביבה (Data-Driven Design), במערכת הנוטונים והגישה אליהם הם דואקאר כריב שולי בקונטקט של האפליקציה: הרכיב המרכזי שנמצא בלב מערכת Soloist הוא דואקאר ה-Domain – הלוגיקה לייצוג צלילים ומנגינות, העיבוד שלהם וכל האלגוריתמים המתוחכמים שיחד מרכיבים את השירותים המרכזיים שהמערכת מספקת – הלחנה אוטומטית של מנגינות. לפיכך עיצוב כל המערכת נעשה סביבת תחום זה של הלחנת המנגינות בעיצוב שמנוהת תחת תחום היעיוס (Domain-Driven Design). באופן סכמטי, נח לתאר את הארץית שמבנה המערכת שנבנתה בעיצוב זה כסדרה של טבעות, כאשר כל טבעה מייצגת שכבה, והשכבות שמייצגות ע"י הטבעות הפנימיות מספקות שירותים לשכבות שמייצגות ע"י הטבעות החיצונית יותר –



נסקרו כעת בקצרה את השכבות המרכזיות המרכיבות את המערכת כפי שהן מתוארות בתרשימים שלעיל, ובפרק הבא נעבור על כל אחת מהשכבות בנפרד ועבור כל שכבה נתאר באופן מפורט את כל רכיבי הפענוח שמרכיבים אותה – מחלקות, מנשכים, אינטראקציות וכו'.

4.2 שכבות המערכת

להלן תקציר של השכבות הלוגיות והפיזיות השונות המרכיבות את המערכת. זהה סקירה מותמצצתת בלבד. בפרקם הבאים כל אחת מהשכבות שליל מתוארת נפרדת בפרט פרטים.

1. שכבה הלוגיקה העסקית של האפליקציה (Domain Business Logic Layer) – שכבה זו מהווה את הלב והמוח של המערכת. היא מכילה את כל הלוגיקה העסקית והאלגוריתמים הדרושים לייצוג, עיבוד והחנה של מנגינות. בכך לבדד את כל הלוגיקה המורכבת הזו ליחידה אוטומטית שאינה תלולה ביתר רכיבי הפיתוח, שכבה זו נבנתה בגישה מוכוונת-שירות (service oriented) : הוגדרו מנשקים חיצוניים פשוטים (high-level) של כל השירותים שכבה זו אחראית לפסק, וכל הלוגיקה למימוש מנשקים אלו כמוסים ומוסתרים מיישומים המשתמשים בהם. בתרשים, הממשק שחווסף את השירותים ליתר השכבות מוצג ע"י שטח הטעבת הסגולה, ואילו המימוש של כל השירותים הללו מוצג ע"י השטח הירוק שבמרכזו התרשים, וכן, לב המערכת הא מימוש הלוגיקה של כל השירותים הללו.

2. שכבה הגישה לנוטונים (Data Access Layer) – שכבה זו אחראית על ניהול הגישה לנוטונים על כל המשתמע מכך – ריכזו מחלקות ה-*Domain Models*, אינטראקציה מול מסד הנתונים, מיפוי בין הטבלאות והרשומות שבבסיס הנתונים לבין הישויות והאובייקטים המיוצגים במחלקות ה-*Domain Model*, ו-Assadפקת שכבת הפעטה מעל הגישה לנוטונים באמצעות Repositories ומנהל ישות המתפרק כ-*Unit Of Work*, מה שמבטיחה אי-תלות של השכבות החיצונית בבסיס הנתונים, וכן אי-תלות בטכנולוגיות ה-*ORM* המממשת את המיפוי שבין הטבלאות למחלקות. כל הגישות לבסיס הנתונים מבוסעות אך ורק באמצעות אמצעות השכבה הזו. אף שכבה אחרת אינה נגישה לבסיס הנתונים ישירות. ארQUITטורתה זו מבטיחה שהחלפת מוצר ה-*ORM* או בסיס הנתונים יכולה להיעשות באופן "שקוף" לשכבות האחרות, כל עוד שכבת הגישה לנוטונים דואגת לשמור את מימוש הממשק הציבורי שלה אל מול ה-*ORM*/מסד נתונים החדש.

3. שכבת התצוגה (Presentation Layer) – שכבה זו אחראית על האינטראקציה מול משתמשי הקצה – הגדרת ממשק משתמש גרפי (GUI), האזנה לקלט המשתמש ונתן מענה לבקשות שלו תוך אינטראקציה עם שכבות הגישה לנוטונים לאחזר ועדכו רשותות וכן שכבת הלוגיקה העסקית של האפליקציה לטובת צricht השירותים שהמשתמש מבקש. בידוד התצוגה מהלוגיקה העסקית ומהגישה לנוטונים מאפשר להציג תצוגות שונות ו春风 aplikציות שונות (מנקודת המבט של המשתמש) המתבססות על השירותים שכבות הלוגיקה והנתונים ומיציאות חוויות משתמש שונות. במימוש הנוכחי הוחלט למשתמש את שכבת התצוגה במכשיר Web-י – בפלטפורמת ASP.NET MVC, אולם הוגדרו גם פרויקטים נוספים עם תצוגה אלטרנטיבית – האחד פרויקט Windows Forms שנבנה כאב-טיפוס למערכת, ומציג ממשק חלוני ביישום שולחני (Desktop Application) מותאים למחשב PC של Windows. השני הוא פרויקט Console Application, החושך ממשק משתמש מינימאלי של קונסול עם מפרש לשורות פקודה. שני פרויקטים אלו מתבססים על שירותים שכבת הלוגיקה העסקית וממחישים אלטרנטיביות שונות למימוש שכבת תצוגה.

השכבות שליל חולקו הן באופן לוגי (לפי מרחב שמות – Namespace) והן באופן פיזי (כל שכבה בפרויקט נפרד, המהווה יחידת Assembly נפרדת). להלן הפרויקטים הפיזיים ומרחבי השמות הלוגיים של כל אחד משלוש השכבות שליל –

#	שכבה	פרויקט	מרחב שמות (Namespace)
<u>1</u>	לוגיקה עסקית (BLL)	CompositionService	CW.Soloist.CompositionService
<u>2</u>	גישה לנוטונים (DAL)	DataAccess	CW.Soloist.DataAccess
<u>3</u>	תצוגה (PL)	WebApplication	CW.Soloist.WebApplication

5 שכבת הלוגיקה העסקית (BLL)

שכבה זו מתחילה את הלב והמה של המערכת, וכוללת את כל הלוגיקה העסקית הנדרשת לייצוג ועיבוד של תווים, ועוד אלגוריתמים להלחנת מגניות חדשות. שכבה זו מוגדרת בשלמותה כספרייה בפרויקט ייודי המהווה יחידת אסambil עצמאית: CompositionService. בהתחם (וע"פ [מוסכמות השמות במערכת למרחבי שם](#)), כל מרחבי השמות (namespaces) השייכים לשכבה זו מושרים תחת מרחב השמות: CW.Soloist.CompositionService.

מטרת העל של שכבת הלוגיקה העסקית היא אספקת שירות להלחנת מגניות. על מנת להשיג מטרת על זו, בוצעה חלוקה ליעדי משנה, שהשגה מלאה שלהם מביאים לימוש מטרת העל. להלן – יעדי המשנה העיקריים של שכבת הלוגיקה העסקית שיחד ממשימים את מטרת העל –

1. **ייצוג ישויות מוסיקליות** – ייצוג של תווים, מרווחים, אקורדים, סולמות, תיבות וכו'.
2. **שימוש לוגיקה אלגוריתמית להלחנת מגניות** – כתיבת אלגוריתמים לעיבוד וניתוח של מגניות, יצירה ועכון של המגניות, הערצת המגניות הנוצרות ועוד, תוך שימוש בישיות המוסיקליות כפי שמיוצגות במערכת.
3. **ניהול אינטראקציה ומיפוי של קבצי MIDI** – קריאת קבצי MIDI, פענוח תוכן קבצי MIDI והמרת אירובי-h-MIDI המפוענחים לשירות המוסיקליות כפי שמיוצגות במערכת, יצירת קבצי MIDI חדשים, עדכון רצoutes בקבצי MIDI קיימים, נגון קבצי-h-MIDI בקבצי שמע, ועוד.
4. **עטיפת הלוגיקה בשירות פשט** – כימוס הלוגיקה המורכבת המוגדרת בשכבה זו ואספקתה כשירות במנשך פשוט וקל לשימוש (high-level) שהיה זמין לשכבות חיצונית ויסתיר את הלוגיקה הפנימית.

בדי להוציא לפועל יעדי-משנה אלו, שכבת הלוגיקה העסקית חולקה לתתי-שכבות, או מודולים – רכיבים לוגיים שונים תחת מרחבי שמות נפרדים, כאשר כל רכיב אחראי על השלמת אחד היעדים השונים. להלן סקירה קצרה של הרכיבים השונים המרכיבים את שכבת הלוגיקה העסקית עם מרחב-השמות הפנימי של כל רכיב (תת-שכבה/מודול) –

#	תת-רכיב בשכבת הלוגיקה העסקית	שם מרחב שמות (Namespace) של הרכיב
1	ניהול האינטראקציה בין הרכיבים השונים של השכבה ועטיפתם במנשך פשוט שיהיה זמין לשירות לשכבות חיצונית	CW.Soloist.CompositionService
2	ייצוג ישויות מהתאוריה המוסיקלית	CW.Soloist.CompositionService.MusicTheory
3	שימוש אלגוריתמים להלחנת מגניות	CW.Soloist.CompositionService.Compositors
4	אינטראקטיה עם קבצי MIDI	CW.Soloist.CompositionService.Midi
5	אוסף עזר של קבועים	CW.Soloist.CompositionService.UtilEnums

בעמודים הבאים מתוארים הרכיבים השונים בפירוט.

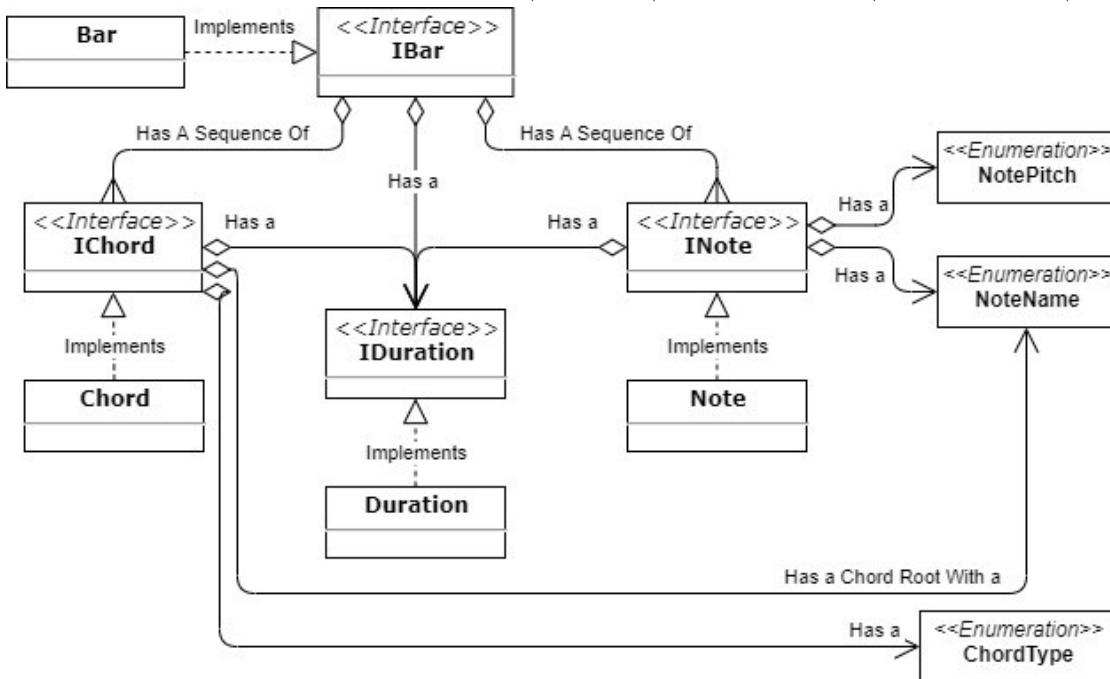
5.1 ייצוג ישויות מוסיקליות (Music Theory)

CW.Soloist.CompositionService.MusicTheory :
כל הישויות מוגדרות במרחב שמות הבא :

להלן רכיבי הפיתוח המוגדרים בתת-שכבה זו עבור הישויות המוסיקליות השונות –

#	תיאור / ישות מוסיקלית	שם הרכיב	סוג רכיב פיתוח
1	ייצוג משך שהייה	IDuration	Interface
2		Duration	Class
3	ייצוג תווים	NotePitch	Enumeration
4		NoteName	Enumeration
5	ייצוג אקורדים	INote	Interface
6		Note	Class
7	ייצוג אקורדים	ChordType	Enumeration
8		IChord	Interface
9		Chord	Class
10	ייצוג תיבות	IBar	Interface
11		Bar	Class
12	ייצוג מרוחקים	PitchInterval	Enumeration
13	יצירת ישויות מוסיקליות	MusicTheoryFactory	Class
14	הרחבות ושירותים כליליים עבור	MusicTheoryExtensions	Class
15	הישויות השונות	MusicTheoryServices	Class

להלן דיאגרמת מחלקה המתארת את הקשרים שבין רכיבי הפיתוח המרכזיים במודול זה –



הערה לחסרי רקע תאורייתי מוסיקלי : מודול זה אחראי כאמור על ייצוג של ישויות מוסיקליות. בתיעוד מוזכרים לא פעם מונחים מוסיקליים כגון תווים, סולמות, וכו'. אולם אין הכרח ברקע מוסיקלי קדימ ב כדי לקבל את ה-"Big Picture" מבחן ה-Design הכללי, אולם כדי להפיך ממשמץ זה את המרב, מומלץ שתהיה הכרות לפחות עם המושגים הבסיסיים. למטרה זו, בסוף מסמך זה מצורף נספח : [רקע תאורייתי של המוסיקה המערבית](#) ב [4 עמודים בלבד](#) המועד לחסרי רקע מוסיקלי ומספק סיכום תמציתי של הידע המוסיקלי הנדרש למסמך זה. על-כן, לחסרי רקע מוסיקלי, מומלץ לעبور על [4 עמודים אלו](#) טרם קראית המשך הפרק, או לחלופין, אם תוכן כדי מעבר על פרטי המימוש נתקלים במושג לא ברור, כדאי פשוט לעבור לנספח שבסוף המסמך באופן נקודתי ב כדי לקבל הבנה על המושג להשלמת התמונה המלאה.

5.1.1 משך שהייה (Duration)

רקע : בהינתן איזושהי מנגינה המורכבת מרצף צלילים, כל צליל מתנגן לאיזושהו פרק זמן סופי. בתאוריה המוסיקלית המערבית, במקום לייצג ולאמוד פרק זמן זה ביחידות זמן אבסולוטיות (כגון שניות או מילישניות), מייצגים את מסכי השהייה של הצלילים השונים באופן רלטיבי ביחס לשאך השהייה הכלול של תיבה שלמה אחת. אם למשל משקל התיבה מוגדר להיות ארבעה (4/4), אזי משך השהייה של חצי תיבה (1/2) משמעו משך שהייה של שני רביעים (שני פעימות עבור משקל של 4/4). ביצוג זה, משך השהייה האבסולוטי של פעימה יחידה אינו רלוונטי לייצוג משך השהייה. משך השהייה האבסולוטי נקבע בהתאם ל-BPM (Beats Per Minute) – מספר הפעימות שנוכחות בדקה אחת, אולם אין זה מעניינים של הצלילים במנגינה, שմבচינתם צריכים רק לדעת מה משך השהייה שלהם ביחס לתיבה הכלילה אותן. לפיכך, בכדי לייצג משך שהייה כל שנדרש הוא שבר – מספר ממשי המייצג את אותו חלק יחסית. למען נוחות בייצוג ובכדי לקבל תמונה מלאה ככל שניתן על משך השהייה באופן כללי כך שיתאים לייצוג משך שהייה גם של צליל בודד אך גם של אקורדים ותיבות, בנוסף למספר המשמעותי המהווה את היחס עצמו, הוחלט למשוך את משך השהייה כך שיכיל גם את המוניה והמכנה שמרכזים את השבר (מנת השבר היא המספר המשמעותי המהווה את היחס).

IDuration Interface 5.1.1.1

המנשך מייצג משך שהייה. להלן החתימות של רכיבי הפיתוח שהוא מגדר –

```
public interface IDuration
{
    // Properties
    byte Numerator { get; set; }
    byte Denominator { get; set; }
    float Fraction { get; }

    // Methods
    IDuration Add(IDuration duration);
    IDuration Subtract(IDuration duration);
    bool IsDenominatorPowerOfTwo();
}
```

תיעוד מלא ומפורט של המאפיינים והמתודות שלעיל ניתן למצוא הן בגוף קוד המקור וכן בדף ה-Wiki של המנשך ב-Github. למען הסדר טוב מובא כאן תקציר התיעוד –

Member Type	Member Name	Description
Property	Numerator (byte)	Number of beats in the duration
Property	Denominator (byte)	Length of the beat in relation to a whole note duration
Property	Fraction (float)	quotient of the division of the duration's numerator by the duration's denominator
Method	Add	Adds the given duration to this duration instance and returns the sum length of them as a new duration
Method	Subtract	Subtracts the given duration from this duration instance and returns the difference length between them as a new duration
Method	IsDenominatorPowerOfTwo	Utility method which determines whether this duration's denominator is a power of 2

Duration Class 5.1.1.2

המחלקה זו מדירה גם בנים, מתודות נוספות והעשרות על אופרטורים של השוואת בין מSCI שהייה. תיעוד מלא ומפורט של כל הרכיבים במחלקה ניתן למצוא חן בגוף קוד המקור וכן [דף Wiki של האפליקציה ב-Github](#). למען הסדר טוב מובא כאן תקציר התיעוד של הרכיבים – (IDuration) – הנוספים השיכים למחלקה עצמה (אלו שאינם מוגדרים במנשך –)

קבועים : הגדרת הערכים הקבועים של מSCI שהייה הנפוצים ביותר –

```
internal const float WholeNoteFraction = 1F;
internal const float HalfNoteFraction = 0.5F;
internal const float QuaterNoteFraction = 0.25F;
internal const float EighthNoteFraction = 0.125F;
internal const float SixteenthNoteFraction = 0.0625F;
internal const float ThirtySecondNoteFraction = 0.03125F;
internal const float TripletEighthNoteFraction = 1F / 12;
internal const float TripletSixteenthNoteFraction = 1F / 24;
internal const byte WholeNoteDenominator = 1;
internal const byte HalfNoteDenominator = 2;
internal const byte QuaterNoteDenominator = 4;
internal const byte EighthNoteDenominator = 8;
internal const byte SixteenthNoteDenominator = 16;
internal const byte ThirtySecondNoteDenominator = 32;
internal const byte TripletEighthNoteDenominator = 12;
internal const byte TripletSixteenthNoteDenominator = 24;
```

מתודות: האחת מחזירה מחרוזת המייצגת משך שהייה, והשנייה ממחשת GCD לפישוט שבר פשוט של משך שהייה ע"י צמצום במכנה משותף מקסימלי –

```
// Returns a string that represents the current duration.
public override string ToString() => $"{Numerator}/{Denominator}";

// Static utility method for returning the GCD of two given natural numbers.
internal static int GreatestCommonDivisor(int a, int b) {...}
```

העשרות אופרטורייט: מחלקה זו מדירה מחדש את האופרטורים הבאים להשוואה בין מSCI שהייה כך שניתן להשוות ביןיהם בקלות : <=,>=,<,>

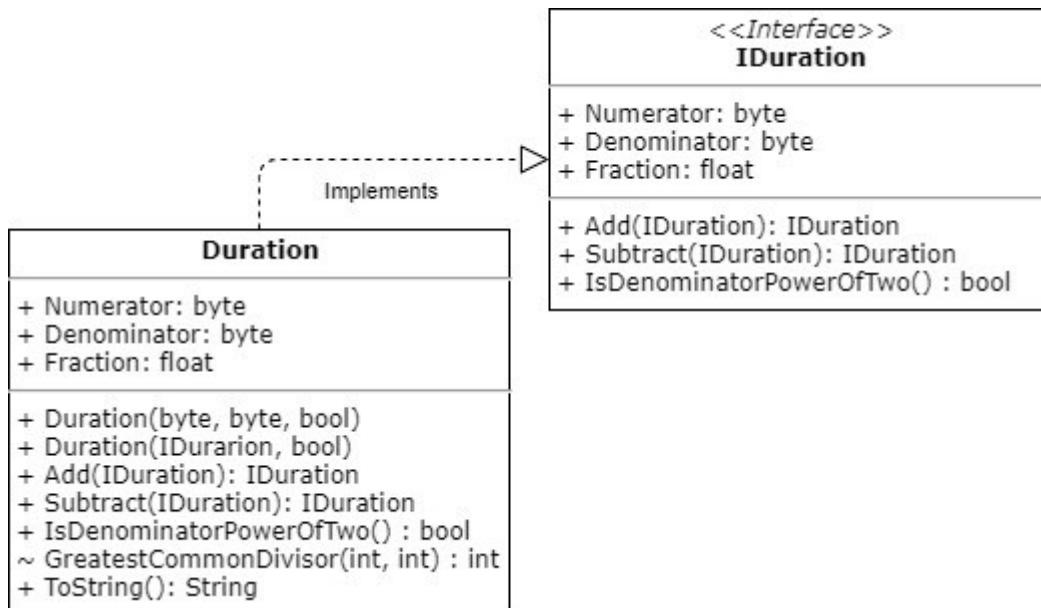
```
public static bool operator <= (Duration duration1, IDuration duration2) {...}
public static bool operator >= (Duration duration1, IDuration duration2) {...}
public static bool operator < (Duration duration1, IDuration duration2) {...}
public static bool operator > (Duration duration1, IDuration duration2) {...}
```

בנייה: אחד מבוסס על מונה ומכנה, שני בניין העתקה, שניהם עם אפשרות לצמצום עם GCD

```
// Constructs a duration based on a numerator and a denominator.
public Duration(byte numerator = 1, byte denominator = 4, bool reduceToLowestTerms = true){...}

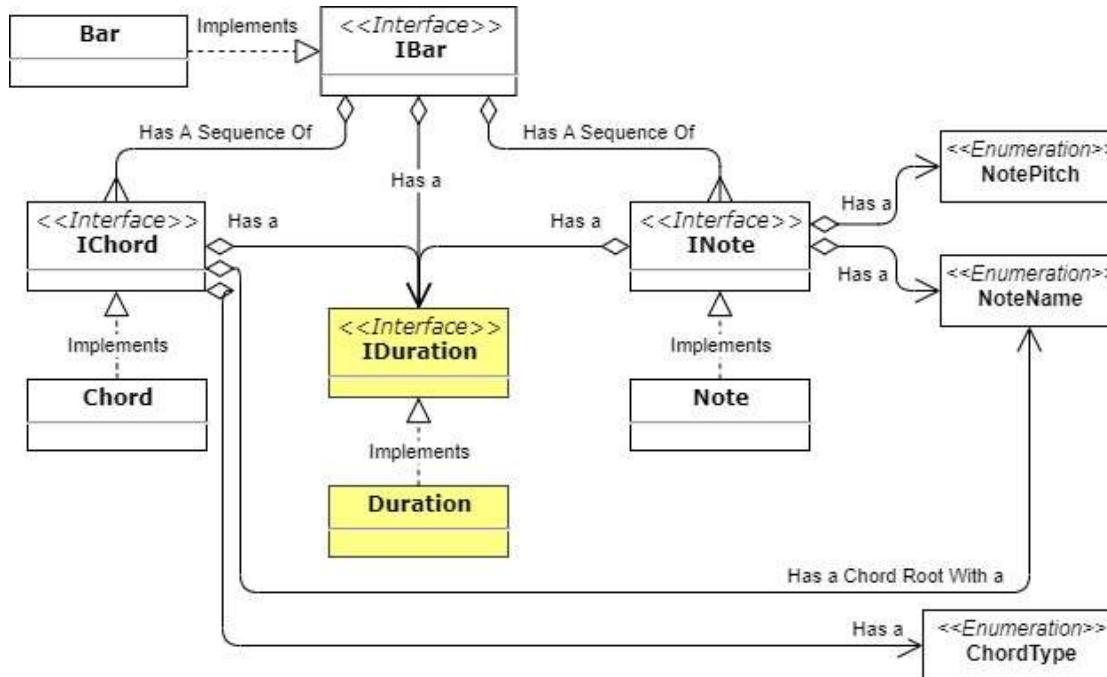
// Constructs a duration based on an existing duration
public Duration(IDuration duration, bool reduceToLowestTerms = true) {...}
```

דיאגרמת Class Diagram עבור המנשך והמחלקה של ייצוג משך שהייה



קשרים עם מחלקות אחרות

המנשך IDuration שמחלקה זו מממשת מהויה יחידה בסיסית המשתתפת בהרכבת יישויות מוסיקליות אחרות: הן לתווים יש משך שהייה, הן לאקורדים והן לתיבות (משך השהייה של ניבעה גדול או שווה משך מסוים הקשורות כל התווים (אקורדים) שהוא מכילה. להלן שוב דיאגרמת המחלקה המתארת את הקשר בין היעילותיות המוסיקליות המרכזיות, עם דגש על הקשר שלחן לישות משך-שהייה –



5.1.2 תוויים (Notes)

רקע : בצד ייציגתו בודד יש צורך בשני מאפיינים : האחד – גובה הצליל, והשני – משך השהייה שלו. משך השהייה מיוצג במערכת ע"י המנשך [IDuration](#) (ראה [ייצוג משך שהייה \(Duration\)](#)). לפיכך נותר להגדיר ייצוג לגובה צליל, שאותו נגדר באטען המכיל ייצוג לכל הצלילים השונים המופיעים ב-MIDI. בנוסף, לטובת קריאות, בהירות וסיווג טקסטואלי לトーויים, נגדר Enumeration לייצוג שמות התוויים השונים (דו, רה, מי, ...) ולבסוף נادر אבני-בניין אלו יחד ונגדר מנשך INote המיציגתו, עם מאפייני משך השהייה וגובה הצליל, ומחלוקת Note הממשת מנשך זה.

NotePitch Enumeration 5.1.2.1

הסטנדרט של MIDI מגדיר מיפוי בין ערכיהם בתחום [0-127] לבין 128 גבהים צלילים שונים, הכוללים בין היתר את כל 88 הצלילים המופיעים ע"י מקלדת פסנתר סטנדרטית. להלן המיפוי המציג עבור כל אחד מגבהי הצלילים את שם התו, את הערך המספרית המופנה לו ב-MIDI ואת התדריות של הצלילים ביחידות הרץ –

Equal Temperament Chart											
Middle C = MIDI Note #60 • Low Piano A = MIDI Note #21 • MIDI Note # range 0-127											
Approximate ideal human hearing range 20-20,000 Hz											
Ratio Between Consecutive Semitones = 12th Root of 2 or $2^{(1/12)}$, approximately 1:1.05946											
Chart Column Format: Pitch MIDI Note Number Cycles Per Second (Hertz)											
C	0	8.175799	C	36	65.406395	C	72	523.251160	C	108	4186.009277
C#	1	8.661957	C#	37	69.295654	C#	73	554.365234	C#	109	4434.921875
D	2	9.177024	D	38	73.416191	D	74	587.329529	D	110	4698.636230
D#	3	9.722718	D#	39	77.781746	D#	75	622.253967	D#	111	4978.031738
E	4	10.300861	E	40	82.406891	E	76	659.255127	E	112	5274.041016
F	5	10.913383	F	41	87.307060	F	77	698.456482	F	113	5587.651855
F#	6	11.562325	F#	42	92.498604	F#	78	739.988831	F#	114	5919.910645
G	7	12.249857	G	43	97.998856	G	79	783.990845	G	115	6271.926758
G#	8	12.978271	G#	44	103.826172	G#	80	830.609375	G#	116	6644.875000
A	9	13.750000	A	45	110.000000	A	81	880.000000	A	117	7040.000000
A#	10	14.567617	A#	46	116.540939	A#	82	932.327515	A#	118	7458.620117
B	11	15.433853	B	47	123.470825	B	83	987.766602	B	119	7902.132812
C	12	16.351599	C	48	130.812790	C	84	1046.502319	C	120	8372.018555
C#	13	17.323914	C#	49	138.591309	C#	85	1108.730469	C#	121	8869.843750
D	14	18.354048	D	50	146.832382	D	86	1174.659058	D	122	9397.272461
D#	15	19.445436	D#	51	155.563492	D#	87	1244.507935	D#	123	9956.063477
E	16	20.601723	E	52	164.813782	E	88	1318.510254	E	124	10548.082031
F	17	21.826765	F	53	174.614120	F	89	1396.912964	F	125	11175.303711
F#	18	23.124651	F#	54	184.997208	F#	90	1479.977661	F#	126	11839.821289
G	19	24.499714	G	55	195.997711	G	91	1567.981689	G	127	12543.853516
G#	20	25.956543	G#	56	207.652344	G#	92	1661.218750	G#	n/a	13289.750000
A	21	27.500000	A	57	220.000000	A	93	1760.000000	A	n/a	14080.000000
A#	22	29.135235	A#	58	233.081879	A#	94	1864.655029	A#	n/a	14917.240234
B	23	30.867706	B	59	246.941650	B	95	1975.533203	B	n/a	15804.265625
C	24	32.703197	C	60	261.625580	C	96	2093.004639	C	n/a	16744.035156
C#	25	34.647827	C#	61	277.182617	C#	97	2217.460938	C#	n/a	17739.687500
D	26	36.708096	D	62	293.664764	D	98	2349.318115	D	n/a	18794.544922
D#	27	38.890873	D#	63	311.126984	D#	99	2489.015869	D#	n/a	19912.126953
E	28	41.203445	E	64	329.627563	E	100	2637.020508	E	n/a	21096.164062
F	29	43.653530	F	65	349.228241	F	101	2793.825928			
F#	30	46.249302	F#	66	369.994415	F#	102	2959.955322			
G	31	48.999428	G	67	391.995422	G	103	3135.963379			
G#	32	51.913086	G#	68	415.304688	G#	104	3322.437500			
A	33	55.000000	A	69	440.000000	A	105	3520.000000			
A#	34	58.270470	A#	70	466.163757	A#	106	3729.310059			
B	35	61.735413	B	71	493.883301	B	107	3951.066406			

מייפוי זה שימושי ביותר, שכן MIDI הוא אחד הסטנדרטים המרכזיים לייצוג תווים וצלילים בתכניות מחשב (אם לא "הסטנדרט" בהא הידיעה) וכן ייצוג גבויו צלילים ע"פ אותן ערכיהם קרייטי לטובת מציאת שפה אחידה ומשותפת עם ספריות חיצונית.

נכון לזמן כתיבת המערכת, מאגר הספריות הסטנדרטיות של פלטפורמת ה-.NET. לא כלליה ספרייה לעובדה מול הסטנדרט של MIDI. עם זאת, ישנו ספריות חיצונית רבות, ובאחד מהן אכן נעשה שימוש לטובת עיבוד ואינטראקציה מול קבצי MIDI, אולם לטובת ייצוג האספект התאורטי המוסיקלי נטו (ייצוג גבוי צלילים), יהיה זה חבל להיות תלויים במימוש ספציפי כזה או אחר של אחת מספריות אלו, ככל שנדרש הוא לייצג תווים. לפיכך התקבלה החלטה פשוטה למשתמש מייפוי שירותים כאוסף קבועים, דהיינו כ-Enumeration. אוסף זה מוגדר

בקובץ [NotePitch.cs](#)

```
public enum NotePitch
{
    // Pitches
    CMinus1 = 0,
    CSharpMinus1 = 1,
    ...
    FSharp9 = 126,
    G9 = 127,

    // Special notes
    HoldNote = 128,
    RestNote = -1
}
```

מוסכמת (קונבנציית) השמות שניתנו לגובה הצלילים הם שם התו מלוה במספר האוקטבה (ע"פ המייפוי ב-MIDI), למשל G9 משמע התו "G" ("סול") באוקטבה התשיעית. עברו צלילים שדורשים סימני התק (הצלילים המנגנים ע"י הקלידים השחורים במקלדת פסנטרה) נבחר לקרוא לשם התו עם סימן התק של דיאז (#). אולם זהו עניין של מוסכמה בלבד. כמובן שהצליל המיוצג ע"י FSharp (פה דיאז) הוא בדיק אוטומטי הצליל המיוצג ע"י GFlat (סול במול), פשוט מטעמי נוחות, הוחלט להידבק לשיטה רישום אחת ונבחרה הריאשונה. השם כאן מילא פחות רלוונטי, מה שקובע בסופו של דבר הוא הערך המספרי שקובע את גובה הצליל שיישלח ל-MIDI, וזהו בסופו של דבר יצא זהה בין אם מייצגים את תוכי ההתק עם דיאזים ובין אם עם במולים.

- בנוסף לערבי גבוי הצלילים המוגדרים ב-MIDI בתחום [127-0], הוגדרו שני ערכים נוספים הדורשים לייצוג תווים "מיוחדים":
1. **HoldNote** – מייצג תוכחת, דהיינותו שאיינו מנגן צליל חדש אלא משמר את גיוון הצליל שקדם לו ובכך למעשה מאריך את משך השהייה שלו, כאשר משך תוספת ההארכה הוא משך השהייה שלתו **HoldNote**.
 2. **RestNote** – מייצגתו השתק של הפסקה ("שקט"). משמעותתו זה היא שלכל אורך משך השהייה שלו משתמש שקט (אף צליל איינו מותגן).

שני תווים מיוחדים אלו אין באמת גובה צליל ממש עצם, אלא רק משך השהייה. כתוצאה לכך, בשיטות הייצוג הנוכחית של תווים שנבחרה בפרויקט, שבה כלתו מרכיב מגובה צליל ומשך השהייה, יוצא שערך גובה הצליל הוא מנון וחסר משמעות. בכך לחסוך את הצורך בהוספה שדה/שדות נוספים שייציגו את סוג התו (האם מדובר בסוגתו שמנגן צליל "אמיתי", או תוכחת /תו השתק), ובהתאם לדעת כיצד יש לפענה את ערך גובה הצליל והאם הוא מנון או לאו, הוחלט פשטוט *"להתלבש"* על מאפיין גובה הצליל ולאחריו ערך מיוחד לסיווג התו במקורה המייחד שאכן מדובר בתו מיוחד. זהה דרך אגב שיטת ייצוג מקובלת (קיים ערכים מיוחדים לתווים השתק ותווי-קשת). למעשה, ההשראה למימוש זה נלקחה ממאמרים המתארים את המימוש של מערכת [GenJam](#) – מערכת המאפשרת מנוגנות סולו בגיאז בזמן אמת בהופעות חיות באינטראקציה עם הנגנים על הבמה, תוך שימוש באלגוריתמים גנטיים ומגר פרזות (מנוגנות קרניות) מוכנות מראש.

NoteName Enumeration 5.1.2.2

בכדי לזהות ולסwoג את התווים השונים ע"פ שם (דו, רה, מי,..., וכו'), הוגדר Enumeration המכיל את שמות כל התווים האפשריים במוסיקת במערכת היטונאלית. נביר כי להבדיל מgebhi הצלילים, שבהם אנחנו מבוחנים בין צלילים המייצגים אותותו לבין אוקטבות שונות (צלילים שהתדריות שלהם בהרץ היא ביחס של 2:1 בין אחד לשני באוקטבות עוקבות) ושורה'כ ישנים 128 אלו ב-B-I-MIDI, במיפוי השמי לתווים אנחנו לא מבוחנים בגבhi צלילים שונים של אותותו, לכן התו C למשל, מייצג את התו השיך לצלילים בכל האוקטבות (בטבלה מעלה מופיע של MIDI אלו הם כל הצלילים המשוגרים תחת C ללא תלות במספר האוקטבה, למעשה כל הערכיהם שמתחלקים ב-12 ללא שארית: 0,12,24 וכו').

אוסף שמות זה שימושי כאשר נדרש לשיך צלילים לתווים בהקשר (context) נתון. עם שמות תווים ומיפוי שלהם לצלילים, ניתן למשל להציג בקלות שגרה שתציג את אוסף הצלילים הרלוננטיים לאקורד מז'ור המושרש בתו דו (C), או אוסף הצלילים השיכים לסולם לה-מינור, כשבור השורש מספיק לציין את שם התו, ולא את כל הגבhai הצלילים האפשריים הדבר קצר דומה למיפוי של DNS בין כתובת IP ל-Hostname. המיפוי השמי אינו הכרחי, אך מאוד כדאי בכדי להפוך את החאים לקלים יותר.

אוסף השמות מוגדר כ-Enumeration בקובץ NoteName.cs. אוסף מכיל את כל שמות התווים, כאשר כלתו מופיעת תחת שלושה ערכים – האחד עבור התו "הנקי" עצמו, השני עבור התו עם סימן התק שבלול (הורדה בחצי-טון) ש"פ המוסכמה מוגדר עם סימנת Flat המשורשת לשם התו, ואחרו חביב השלישי, התו עם סימן התק של דייז (העלאה בחצי-טון), ש"פ המוסכמה מוגדר עם סימנת Sharp המשורשת לשם התו.

שמות התווים האפשריים הם כדלקמן –

שם התו בכתב עברית	שם התו בכתב לטיני
לה	A
סי	B
דו	C
רה	D
מי	E
פה	F
סול	G

לפייך באוסף הקבועים מוגדרים השמות A,B,C,D,E,F,G וכן הלאה עד ל-Sharp, Flat, BFlat, ASharp, AFlat, BSharp, וכו' הלאה עד ל-G, כאשר כלתו מופיעת כאמור בשלוש ורייציות: "נקוי", עם תוספת "Flat" לסימון בלול, ועם תוספת "Sharp" לסימון דייז.

אוסף זה למעשה מייצג את שמות התווים האפשריים במערכת Soloist ומשתתף במנשך באופן רוחבי בשכבת הלוגיקה העסקית, הן במסגרת הייצוג הישיות המוסיקליות (למשל בתור מאפייןתו השורש של אקורדים וסולמות) והן בהמרה בין ייצוג שונים של תווים מספריות חייזניות, אלגוריתמי הלחנה ועוד.

[INote Interface](#) 5.1.2.3

המנשך [INote](#) מייצגתו. תיעוד מלא ומפורט של המאפיינים שליל ניתן למצוא הנו בגוף קוד המקור וכן בדף ה-Wiki-של המנשך, המכיל קישורים לטיפוסי המאפיינים השונים עם התיעוד שלהם. למען הסדר טוב מובא כאן תקציר התיעוד לצד ההגדרות עצמן –

```
NoteName? Name { get; } // The note's name.  
NotePitch Pitch { get; } // The note's absolute pitch.  
IDuration Duration { get; } // The note's duration.
```

המנשך מורכב מישויות שסקרנו קודם לכן – [משך שהייה](#), [גובה צליל](#), [שם تو](#).

[Note Class](#) 5.1.2.4

מחלקה [Note](#) מימושת את המנשך [INote](#). מעבר למאפיינים והמתודות במנשך, מחלוקת זו מגדריה גם בנאים, קבועים, ומתודות לאחוזה שם הנו ומחרוות לתיאור הינו. תיעוד מלא ומפורט של כל הרכיבים במחלוקת ניתן למצוא הנו בגוף הקוד המקור וכן בדף ה-Wiki-של [האפליקציה](#). למען הסדר טוב מובא כאן תקציר התיעוד של הרכיבים הנוספים השויכים למחלוקת עצמה (אלו שאינם חלק מהמנשך [INote](#)) –

בנייה: בניאי מבוסס צליל ומשך שהייה, צליל ומוונה/מכנה, ובנאי העתקה –

```
// Constructs a new note instance based on a given pitch and duration.  
public Note(NotePitch pitch, IDuration duration) {...}  
  
// Constructs a new note instance based on a given pitch, and a duration which  
// is composed with the quotient of the numerator divided by the denominator.  
public Note(NotePitch pitch, byte numerator, byte denominator) {...}  
  
// Deep Copy Constructor: constructs a new note based on an existing note.  
public Note(NotePitch pitch) {...}
```

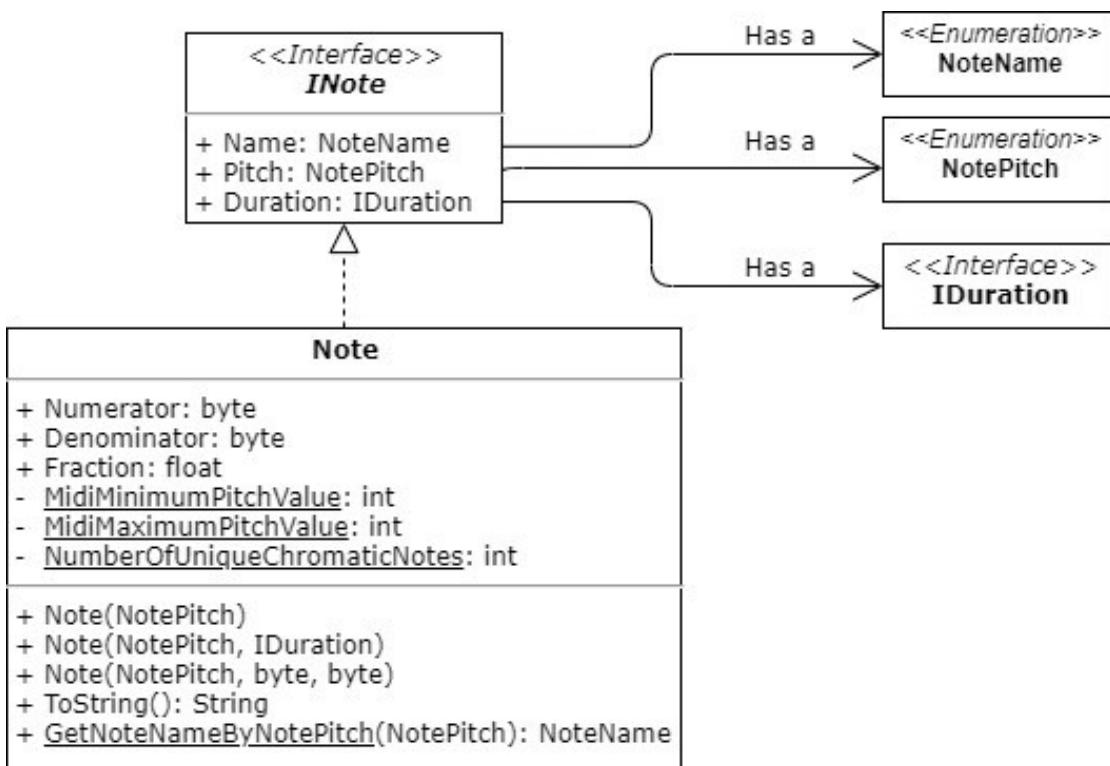
מתודות: ייצוגתו ע"י מחרוזת ואחיזור שם תו לפי גובה הצליל –

```
// Returns a string representing the note's instance state.  
public override string ToString() {...}  
  
// Returns the note name which corresponds to the given note pitch.  
private static NoteName? GetNoteNameByNotePitch(NotePitch notePitch) {...}
```

קבועים: קבועים אלו משמשים בבדיקות תקינות, ושגורות עזר במניפולציות על תווים –

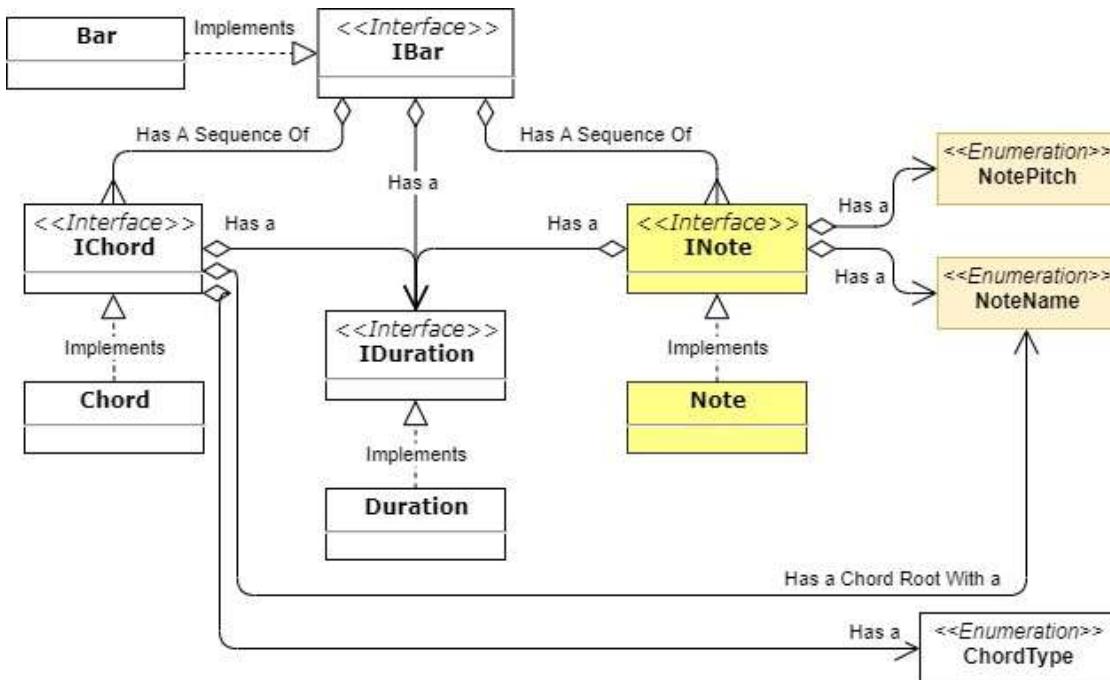
```
private const int NumberOfUniqueChromaticNotes = 12;  
private const int MidiMinimumPitchValue = 0;  
private const int MidiMaximumPitchValue = 127;
```

דיאגרמת Class Diagram עבור היישוות לייצוג תווים – Class Diagram for Music Theory Representation



קשרים עם מחלקות אחרות – Relationships with other classes

רכיבי הייצוג של תווים בפרק זה נמצאים בשימוש באופן רוחבי בשכבה הולוגית העסקית, שכן התווים מהווים את אבני הבניין של המנגינה וכל אלגוריתמי הלחנה בסופו של דבר מבצעים מניפולציות על תווים אלו. להלן שוב דיאגרמות המחלקה של היישוות המוסיקליות המרכזיות בדגש על הקשר שלחן עם היישוות המיציגות תווים שהוצגו בסעיף זה –



5.1.3 אקורדים (Chords)

רקע: אקורד הוא אוסף של שלושה צלילים (שונים) או יותר. בראש כל אקורד מוגדר שורש האקורד, שהוא התו הנמוך ביותר באקורד. יתר הצלילים באקורדים מוגדרים ע"פ רצף מרוחקים קבוע מראש מהשורש. רצף מרוחקים קבוע זה מוגדר ע"פ סוג האקורד (מז'ור, מינור, וכד'). למשל: אקורד משולש מז'ורי מגדיר מרוחק של ארבעה חצאי טוניים מהשורש לצליל השני באקורד ולאחר מכן מרוחק של שלושה חצאי טוניים בין הצליל השני באקורד לצליל השלישי.

ביצירה מוסיקלית נתונה, האקורדים פרוסים לאורך היצירה וקובעים את המהלך החרמוני שלה. אקורדים אלו מתגנים לצד או "אחוריו" המנגינה, ומספקים לה בעצם את התשתית והליווי. כדי להגדיר את המיפוי והסנכרון בין צלילים במנגינה לבין צלילי האקורדים המלוויים אותם, גם לאקורדים יש מSCI שהייה בדיק כפי שתותווים מתגנים יש משך שהייה. לפיכך, כדי ליציג אקורדים נדרשים שלושה מאפיינים – התו הנמצא בשורש האקורד, סוג האקורד, שקובע את רצף המרווחים מהשורש למגדירים את יתר הצלילים שבאקורד, וכן משך שהייה, שקובע את זמן הנגינה של כל אקורד ביחס לתיבה שהוא מתגן ביצירה מוסיקלית נתונה.

ChordType Enumeration 5.1.3.1

אוסף הקבועים ב- [מגדיר את סוגי האקורדים הנתמכים ע"י המערכת](#) –

```
public enum ChordType { ... }
```

הגדרת סוגי האקורדים בסט קבועים מוגדר היטב שכזה מאפשר להגדיר היטב את האקורדים ע"י הגדרת שירותים נלווהו שמאחזרים את הצלילים/תוויים הרלוונטיים בהתאם לסוג האקורד, וכן סט סוגי האקורדים מגדיר פורמט לשוגי האקורדים שימוששי קצה יכולם לציין בקבץ קלט שהם מעלים למערכת.

IChord Interface 5.1.3.2

המנשך [מייצג אקורד](#). אקורד מאופיין ע"י ישוות שסקרנו קודם לכן: [שם התו של שורש האקורד, סוג האקורד ומשך שהייה](#) –

```
NoteName ChordRoot { get; } // The chord's root note name.  
ChordType ChordType { get; } // The chord's type(structure).  
IDuration Duration { get; set; } // The chord's duration.
```

המנשך מגדיר שני זוגות של מתודות עוזר לאחיזור אוספי תווים/צלילים ע"פ סוג האקורד: האחד מחזיר את הצלילים של תווים מבנה האקורד עצמו, והשני מחזיר אוסף רחב יותר של צלילים מסוימים שמומפה מול סוג האקורד (המיפוי עצמו ממומש בשירות נפרד מחוץ למנשך). המתודות מאפשרות להעביר חסמים תחתונים ועליונים למנעד הצלילים שיחזור באוסף, ומוגדרים בשני העמיסות, הראשונה מקבלת את קצוות המנעד באוקטבות, והשנייה מקבלת את קצוות המנעד – [לפי גבהי צלילים קונקרטיים](#) –

```
// Returns an arpeggio note sequence of the given chord's.  
IEnumerable<NotePitch> GetArpeggioNotes(int minOctave, int maxOctave);  
IEnumerable<NotePitch> GetArpeggioNotes(NotePitch minPitch, NotePitch  
maxPitch);  
  
// Returns a note sequence from a scale that is mapped to the given chord  
// type. IEnumerable<NotePitch> GetScaleNotes(int minOctave, int maxOctave);  
IEnumerable<NotePitch> GetScaleNotes(NotePitch minPitch, NotePitch maxPitch);
```

תיעוד מלא ומפורט של המאפיינים שליעיל ניתן למצוא להן בגוף קוד המקור וכן בדף ה-Wiki של [המנשך בספריית האפליקציה ב-Github](#).

Chord Class 5.1.3.3

המחלקה Chord מimplements את הממשק IChord. מלבד שימוש המאפיינים והmethodות של הממשק, ההגדירות הנוספות במחלקה כוללות רק בניין וmethod לאחזר מחרוזת המייצגת את האקורד.

תיעוד מלא ומפורט של כל הרכיבים במחלקה ניתן למצוא להנוף קוד המקור וכן [דף ה-Github של האפליקציה ב-Github](#). למען הסדר טוב מובא כאן תקציר התיעוד של הרכיבים הנוספים השiciasים למחלקה עצמה (אלו שאינם חלק מהמנשך (IChord)).

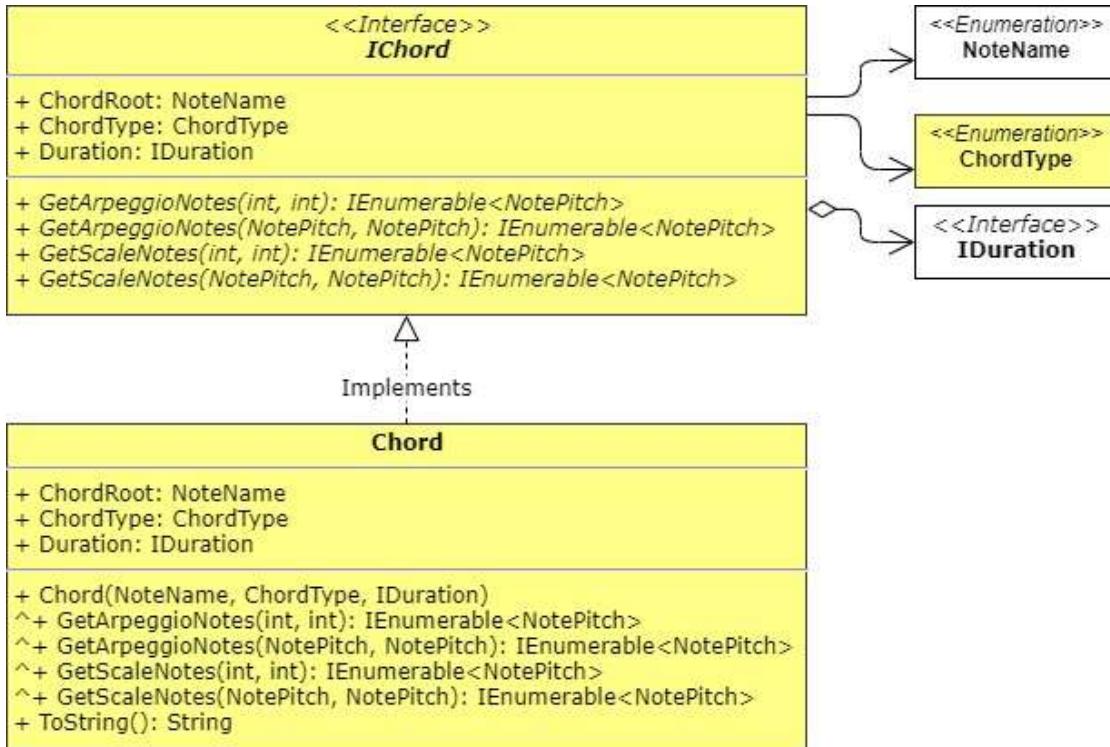
– בנייה: המחלקה מדירה בנאי יחיד ביוצר אקורד על סמךתו שורש, סוג אקורד ומשך-שהיה –

```
// Constructs a new chord instance with the given root, chord type & duration.
public Chord(NoteName root, ChordType type, IDuration duration)
```

– מתודות: המחלקה מדירה מימוש של ToString לאחזר מחרוזת המייצגת אקורד –

```
public override string ToString() =>
    $"{{Root={ChordRoot}; ChordType={ChordType}; Duration={Duration}}}"
```

דיגרמת מחלקה: להלן דיאגרמת מחלקה של הישויות המייצגות את האקורד שהוצגו בסעיף זה – עם הקשרים ביניהם –



5.1.4 תיבות (Bars)

רקע: יצירות מוסיקליות מחולקות לתיבות מסוימות בסגמנטציה וסדר ברצף הצלילים של היצירה. לכל תיבה מוגדר משך שהיא, בדיקת כמו יש לtones ולאקורדים משך שהיא. בהקשר של תיבה, משך שהיא זה נקרא גם משקל, או חותמת זמן (Time signature). לרוב המשקל אינו משתנה בין תיבה לתיבה ונשאר קבוע (זהה) לכל אורך היצירה, אולם המשקל יכול גם להשתנות בין תיבה לתיבה.

כל תיבה מכילה אוסף צלילים שמרכיב את המנגינה ואוסף אקורדים שמרכיב את המהלך הARMONIC של הצלילים הללו. תיבה יכולה גם להיות ריקה, ולהכיל רק אקורדים או טווי הפסקה, במקרה שבו יש מקטע של הפוגה במנגינה או בליווי ביצירה. משך השהייה הכלול של רצף אקורדים ו/או רצף טוים בתיבה נתונה לא יכול לחזור ממשקל התיבה. במקרה שבו משך זמן השהייה של TWO מתגונן חוצה שתי תיבות או יותר, המעבר בכל תיבה עוקבת מסומן עם TWO קשת-חיבור, שמאיריך את משך השהייה, כך שבગוף כל תיבה אכן אין חריגה מהמשקל שלה.

בכדי ליצג תיבה, נדרשים איפה שלושה מאפיינים: חותמת הזמן של התיבה, אוסף הטוים שהיא מכילה (טווי המנגינה), ואוסף האקורדים שבתיבה. כמו כן, לאחר שהאקורדים מלאוים את טווי המנגינה, יש צורך במנגנון של SNCRONIZERS ב כדי לדעת בנקודת זמן נתונה שבה מתגונן TWO איזה אקורד מתגונן מאחריו בליווי ולהפוך: איזה טוים מתגונם באותו מחרוז זמן של אקורד נתון בתיבה. SNCRONIZERS זה נדרש ב כדי להתאים טוים לאקורדים בהלחנת מנגינות חדשות.

IBar Interface 5.1.4.1

המנשך [IBar](#) מייצג תיבה ע"פ המאפיינים שתוארו לעיל –

```
IDuration TimeSignature { get; } // The bar's duration.
IList<IChord> Chords { get; } // Chords which represent the bar's harmony.
IList<INote> Notes { get; set; } // Notes which represent the bar's melody. {
```

להלן המethodות שתוארו לעיל המוגדרות במנשך –

```
// Returns the chords that are played in parallel to given note.
IList<INote> GetOverlappingNotesForChord(int chordIndex, out IList<int> chordNotesIndices);
IList<INote> GetOverlappingNotesForChord(IChord chord, out IList<int> chordNotesIndices);

// Returns the notes that are played in parallel to given chord.
IList<IChord> GetOverlappingChordsForNote(int noteIndex);
```

תיעוד מלא ומפורט של המאפיינים והmethodות שלעיל ניתן למצוא כאן בגוף קוד המקור וכן ב-[DEV-Wiki-של המנשך](#), המכיל קישורים לרכיבים השונים עם התיעוד שלהם.

Bar Class 5.1.4.2

המחלקה Bar מimplements את הממשק IBar. מעבר למימוש המאפיינים ומethods של הממשק, מחלקה זו מדירה רק לבנים ומmethod שאליה מחרוזת המייצגת את התיבה.

תיעוד מלא ומפורט של כל הרכיבים במחלקה ניתן למצוא להנוף הקוד המקורי וכן [דף ה-Wiki של האפליקציה](#). למען הסדר טוב מובא כאן תקציר התיעוד של הרכיבים הנוספים השינויים למחילה עצמה (אלו שאינם חלק מהמנשך IBar).

שדות: המחלקה מדירה שני שדות פרטיים הנזרים מהמבנה והמכנה של משך השהייה – ומגדירים את המשקל של התיבה: מספר הפעימות בתיבה ויחידת המשקל של פעימה בודדת –

```
private byte _beatsPerBar; {...} // Number of beats in the bar.
private byte _beatsDuration; // Duration of a single beat in the bar.
```

– **בנייה:** המחלקה Bar מדירה מגוון בנאים כדלקמן –

```
// Initializes a bar with a default time signature of 4/4, an empty list of
// chords and an empty list of notes.
public Bar() {...}

// Constructs an empty bar based on a given time signature.
public Bar(IDuration timeSignature) {...}

// Constructs an empty bar based on a given time signature & chord progression.
public Bar(IDuration timeSignature, IList<IChord> chords) {...}

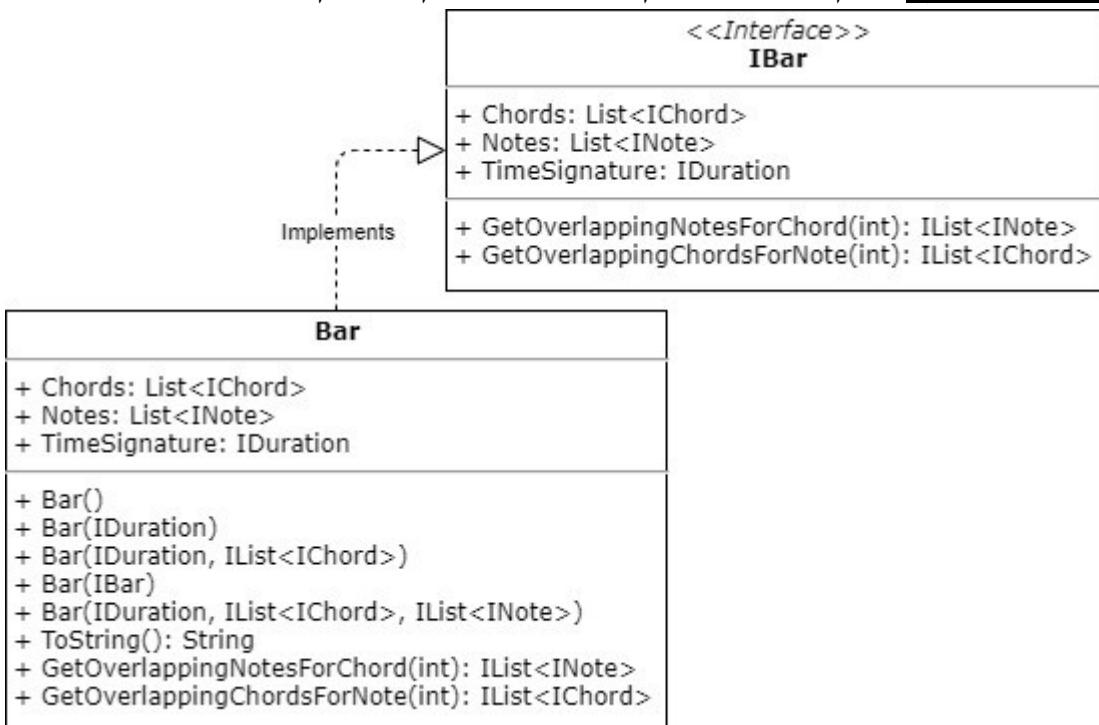
Constructs empty bar based on given time signature, chords, & melody notes.
public Bar(IDuration timeSignature, IList<IChord> chords, IList<INote> notes)

// Copy constructor: notes & durations deep copy, chords is shallow copy.
public Bar(IBar bar) {...}
```

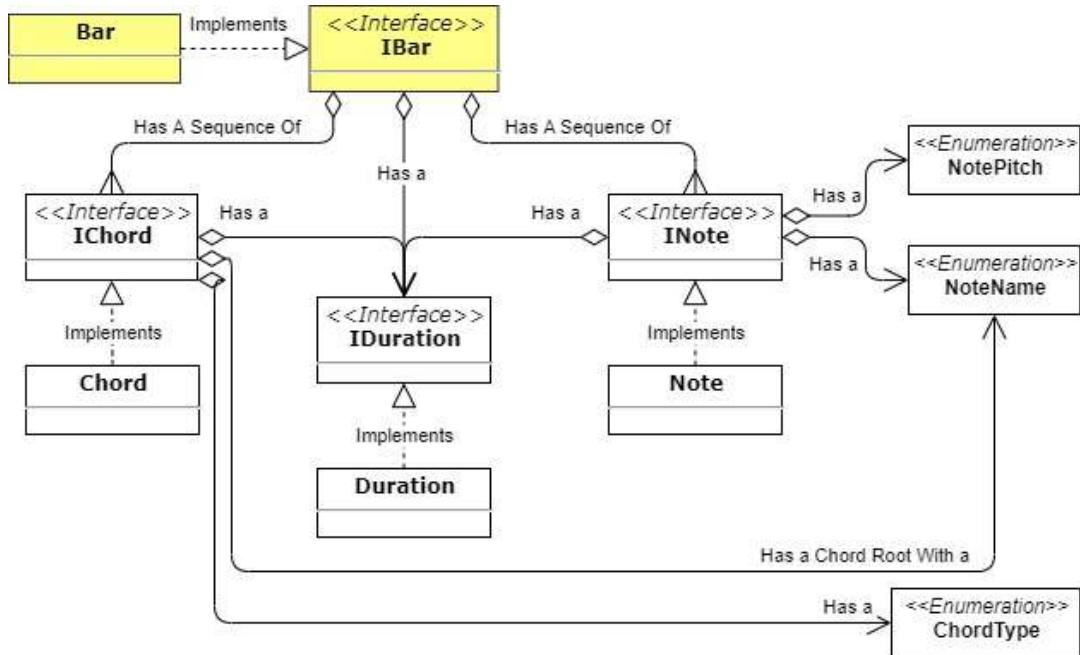
– **מетодות:** המחלקה מדירה מימוש של ToString לאחזר מחרוזת המייצגת תיבה –

```
public override string ToString() {...} // Returns string representation of the bar.
```

– **דיאגרמת מחלקה:** להלן דיאגרמת מחלקה של רכיבי המنشך והמחלקה המייצגים תיבה –



להלן דיאגרמת מחלקה של רכיבי הממשק והמחלקה המייצגים תיבת והקשר שליהן בكونטקסט
עם יתר רכיבי הפיתוח של היישויות המוסיקליות –



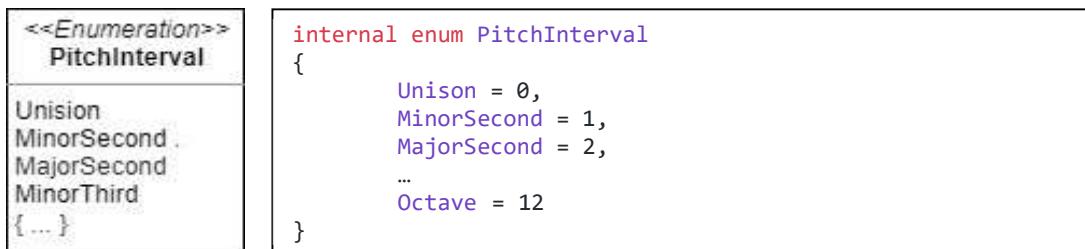
5.1.5 מרווחים (Intervals)

רעיון: מרוחה הוא מרווח בין גביה צליליים, הנמדד בחצאי-טוניים, זהו המרווח הקטן ביותר במוזיקה הערבית – המרווח בין כל שני קלידים בפסנתר (וכן כל שני סרגיגים בצוואר גיטרה). למרוחחים יש משמעות רבה בהלחנת מנגינות – למשל, מרוחה גדולה במיוחד גורם לקפיצה לkiezoniya מציל גבוה לנמוך (או להיפך), מרוחחים מסוימים נוטים להישמע זכרים (מתוקים) בكونטקסט מסוים בעוד שאחרים יוצרים מותח ודיסוננס. לפיכך, יש צורך ביצוג נאות של מרוחחים לטובת אלגוריתמי הלחנה.

מאחר שהתוויים כבר מוגדרים היטב, כל שנדרש לטובת המימוש הוא הגדרת אוסף קבועים עבור המרווחים השונים, כאשר כל קבוע מקודד את המרחוק בחצאי טוניים שהוא מייצג –

PitchInterval Enumeration 5.1.5.1

אוסף זה נמצא בשימוש עיקרי ידי אלגוריתמי הלחנה, משתמשים במרוחחים ב כדי לאמוד ממדדים שונים באשר טוב המעברים בין צליל אחד למשנהו (למשל סיוג מרוחה מהלך ממנו המעבר הוא חד מיידי) וכן בחישובים ארכיטקטוניים של התווים השכנים תוך שימוש במרוחחים.



5.1.6 מחלקה מפעלי לשויות מוסיקליות (MusicTheoryFactory)

מחלקה זו היא מחלקה שירות סטטית המספקת מתודות שירות ליצירת מופעים של מחלקות קונקרטיות שמשמשים את מנשיки היישויות המוסיקליות השונות בתת-שכבה זו של התאוריה המוסיקלית – משך שהיא ([IDuration](#)), צו ([INote](#)), אקורד ([IChord](#)) וטיבה ([IBar](#)), ומכאן שם המחלקה (על שם דפוס העיצוב [FactoryMethod](#)).

מחלקה כוללת ארבעה קטגוריות של מתודות יצירה שונות – קטgorיה אחת כנגד כל ישות מוסיקלית, כאשר בכל קטgorיה המתודות הן בעלות אותו שם ונבדלות רק בסט הארגומנטים שהן.

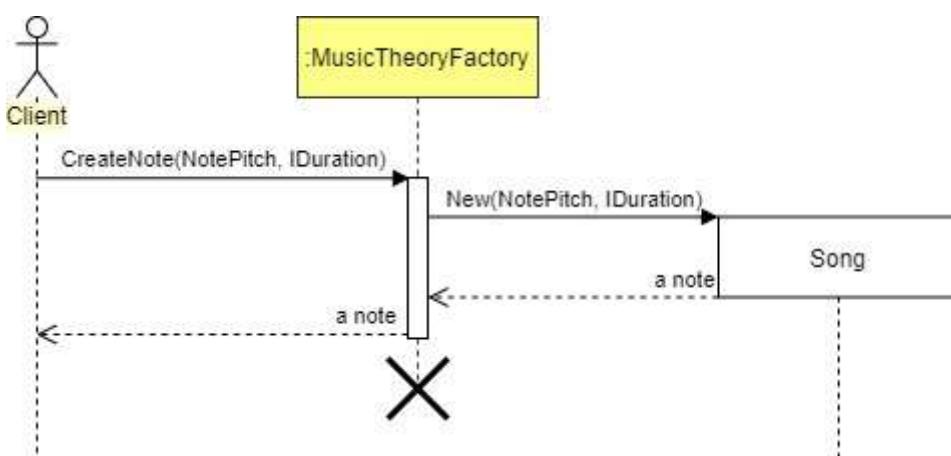
המוסכמה (קונבנצייה) של שמות המתודות במחלקה זו היא "`Create<Entity>`", כאשר `<Entity>` מוחז בשם הישות (ללא אות הראשית I) ממשם ה-Interface I. למשל עבור הממשק [INote](#) המיצג ישות של צו מוסיקלי, המחלקה תגדיר מתודות יצירה בשם "`CreateNote`". להלן היישויות המנוולות כרגע תחת מחלקה זו –

#	Entity	Method Name
1	IDuration	<code>CreateDuration</code>
2	INote	<code>CreateNote</code>
3	IChord	<code>CreateChord</code>
4	IBar	<code>CreateBar</code>

המודיבציה מתחמי מחלקה זו היא לשבור את התלות של קליניטים בימוש ספציפי של ישות מוסיקלית כזו או אחרת, ולהשאיר תלות רק באבstarטרכיה, דהיינו רק במנשך עצמו. קליניטים המעניינים במופע של ישות מוסיקלית יפנו למחלקה [MusicTheoryFactory](#) בבקשת לקבלת מופע של המنشך, והמחלקה תהיה אחראית לספק מופע שכזה, וכל זאת בלי שהקליניטים צריכים להכיר את המחלקות שספקות את המنشך בפועל מתחמי הקליעים. זהו בעצם מימוש של עיקנון [Dependency Inversion Principle](#).

נכון לעכשו, המימוש של ה-Factory כולל פשט הפניה לבנאים של המחלקות המוגדרות בתת-שכבה זו המשמשות מנסחים אלו, שכן זהו המימוש היחיד המוגדר בעת המערכת, אולם אם בעבר יהיה צורך בהחלפה של מימושי ברירת מחדל אלו במימושים מסוכלים יותר, מלבד הוספת החתימות החדשנות במנסחים עצמים (במידת הצורך) כל שיידרש הוא לבצע את ההחלפה במימוש הפנימי של מתודות ה-Factory, כמובן, לקליניטים עצמים ההחלפה תהיה "שкопה", הם ימשיכו לפנותו לרגיל אל ה-Factory בבקשת מופעים של ישויות מוסיקליות ויקבלו מופעים של המנסחים שלהם, מבלי לדעת מהי המחלקה הקונקרטית שספקת את המימוש.

להלן המחשה של אופן העבודה של קליניטים עם מחלקה ה-Factory. בדוגמה זו הקליניטים מבקשים מה-Factory מופע של [INote](#), וזהו יוצרת מופע של המחלקה [Note](#) הממששת את המنشך [INote](#) ע"י קריאה ישירה לבנאי של המחלקה, ומהזירה את התוצאה לקליניט. מנוקדת המבט של הקליניט, הוא ביחס [INote](#) וקיים את מבווקשו. אין זה מעניינו איך מחלקה ספציפית עומדת מתחמי המימוש, כך שבעתיד, אם יוחלף המימוש של [Note](#) במחלקה אחרת, הדבר יהיה שקוף לקליניטים העובדים ישירות מול המنشך [INote](#) –



5.1.7 הרחבות ושירותים נוספים

רקע: מעבר למשקים, מחלקות ואוסףים קבועים שהוצעו לעיל, המיצגים את הישיות המוסיקליות, הוגדרו בנוסף שתי מחלקות סטטיות (מחלקות שירות) הכוללות הרחבות ושירותי עזר סביר הישיות המוסיקליות השונות – [MusicTheoryExtensions](#) ו- [MusicTheoryServices](#).

MusicTheoryExtensions 5.1.7.1

מחלקה זו כשמה כן היא, אוסף של כל ה-Extension Methods עבור הישיות המוסיקליות השונות המספקות שירותים משלימים לשימושיים בעיקר בעבר אלגוריתמי הלחנה. מחלוקת זו מכילה אך ורק Extension Methods ללא מתודות סטטיות נוספות.

שאלה מתבקשת היא מדוע שירותים אלו הוגדרו במחלקה נפרדת זו כמתודות סטטיות ולא במשקים והמחלקות של הישיות המוסיקליות עצמן כמתודות מופע ישירות. ובכן, הסיבה המרכזית שהשירותים לא הוגדרו במשקים היא מאחר השירותים אלו רלוונטיים בעיקר לאלגוריתמי הלחנה, ובkonkretst הרחב יותר של יישיות מוסיקליות שלאו דוקא עוסקת בהלחנה אלגוריתמית, שירותים אלו "יזהמו" את החזושים השונים המתארים את הפונקציונליות המצופה מכל ישות וישות, ללא תלות בהלחנה. הסיבה שמתודות אלו לא הוגדרו ישירות במחלקות כמיומשים מחוץ למנشك היא שחלק מהשירותים ממילא ניתנים לאוסף של יישויות ולא על ישות בודדת ולכן ממילא יש להגדירם כמתודה סטטית (מתודה Extension גם היא סוג של מתודה סטטית מבחינה התchapiriyah שלה), וסיבה נוספת היא ריצוזיות – יש יתרון בריצוע כל השירותים הנלויים הלו במחלקה אחת, כך שאם בעתיד יוחלף שימוש של אחד המנסקים של הישיות המוסיקליות במימוש אחר/חדש ע"י מחלוקת אחרת, השירותים הנלויים שסופקו ע"י המחלוקת המרכזית לא "ילכו לאיבוד" וימשיכו לתת שירות ללא תלות במימוש הספציפי ע"י מחלוקת כזו או אחרת.

להלן המתודות המספקות את השירותים המרכזים במחלקה זו –

Method	The Subject Entity	Description
GetAllPitches	IEnumerable<IBar>	Returns a flattened sequence of all note in the given bar sequence / bar.
GetAllPitches	IBar	
IsOffBeatNote	INote	Return indication whether the given note in the given context is played on an offbeat or not.
IsOffBeatNote	IBar	
GetDurationInContext	INote	Returns the overall length of the given note in the given note sequence context, which might include hold notes which may hold the given note's duration and thereby making it last longer.
GetPredecessorNote	IEnumerable<IBar>	Returns the preceding note of a given note in the given note sequence.
GetSuccessorNote	IEnumerable<IBar>	Returns the succeeding note of a given note in the given note sequence.
ReduceFraction ToLowestTerms	IDuration	Returns a simplified duration by reducing the numerator and denominator by their greatest common divisor (gcd), for example reduce 4/8 to 1/2.

MusicTheoryServices 5.1.7.1

מחלקה זו מכילה כל שירותים משלים או נוספים הנוגע לשויות המוסיקליות שאינו מוגדר ישירות במנשכים ובמחלקות של הishiות עצמן וכן אינו מוגדר כמתודת Extension של ישיות אלו או של אוסףים שלחו. שירותים אלו כוללים השלמת פונקציונליות קרייטית שחלק מהמחלקות צורכות ולמעשה תלויות בו לטובתימוש המנסחים.

המוטיבציה הראשונית להגדרת שירותים אלו במחלקה ייודית נפרדת היא שחלק מהמיימושים הנוכחיים שלהם תלויים בספריותן בלבד. מตוך רצון שלibilit המודול של התאוריה המוסיקלית תהיה עצמאית ונקייה מטלויות ישירות בספריותן בלבד, כל האינטראקציה של ישיות התאוריה המוסיקלית התלויה בספריות חיצוניתן בלבד ג' נגרעה מتوزע המחלקות המייצגות את הishiות עצמן ורוכזה במחלקה זו. מחלוקת זו בעצם מהווע מעין חוץ המשפע מהריעונות של דפוסי העיצוב של Adapter ו-Bridge, שקווטע את ה-coupling של מחלוקת הליבה מהספריות החיצונית. למעשה, כל יתר המחלקות והמנשכים של ישיות התאוריה המוסיקלית כלל לא מכירם את הספריותן בלבד. אלו מהן שכן נדרשים בהשלמת השירותים הנוספים פונמים בבקשת מחלוקת זו שאחריה לספק אותם בעצמה תוך שימוש פנימי ע"פ הצורך בשירותי עוזר של הספריות החיצונית, באופן שסקוף לשויות הליבה שימושים בשירותים אלו.

כ舍ל הקוד הנוגע לשויות המוסיקליות של הספריות החיצונית מרכז במחלקה אחת, יהיה קל מאוד בהמשך לבדוק את התלוויות בספריות אלו ולהחליף אותן ע"פ הצורך בספריות אחרות או אף במימוש עצמי. למעשה התלות הנוכחיות היא די דלה, ועם טיפה עבודה אפשר למש בקהל את השירותים הללו עבור ישויות התאוריה המוסיקלית, ואחד מקווי המחשבה בניסיון הגיע לע-decoupling היה באמצעות שימוש השירותים אלו באופן עצמאי מ-'scratch', אולם במחשبة נספת הגיעה התובנה שדווקא יש לעשות reuse במה שיש ולא להמציא את הגלגל מחדש רק בשביל אי-תלוות, לכן כן הוחלט לעשות שימוש בספריות הקיימים אך תוך בידולם ורכיבום למחלוקת ייודית, שתבצע המרות דרישות בין הייצוגים של ישיות בספרייה האפליקציה ובין הייצוג המקביל שלן בספריותן בלבד, ותהוו מעין חוץ-מתחם שובר את התלוות של הישיות בקוד מספריות חיצונית.

להלן המתודות המספקות את השירותים המרכזים במחלקה זו –

Method	Description
GetNoteName	Returns a note name for a given note pitch.
ConvertToInternalNoteName	Converts a note name from its external third-party library representation into its equivalent internal representation note name and vice-versa (from external to internal). These are required for using the external third-party services.
ConvertToExternalNoteName	
GetNotes	Returns a note sequence that sounds good under the given chord, according to the mapping source – either the chord structure or a scale which is mapped against the given chord.
DurationArithmetic	Returns a new duration instance which is the result of the given arithmetic operation, and the two is given duration operands.

בנוסף למethodות אלו המחלוקת מגדרה גם קבוע עוזר SemitonesInOctave – העותף את "מספר הקסטן" 12, שמייצג את מספר הצלילים בסולם הדיאטוני המערבי, וה-Enum **ArithmeticOperation** המכיל קבועים עבור הפעולות האריתמטיות המוגדרות על אופרנדים מסוג משך שהייה (IDuration).

5.2 תת-שכבה שירותים MIDI (MIDI)

CW.Soloist.CompositionService.Midi

מרחב שמות (Namespaces) רלוונטי:

5.2.1 רקע – פרוטוקול MIDI וקובצי SMF

- **פרוטוקול MIDI:** MIDI (Musical Instrument Digital Interface) הוא פרוטוקול לשיתוף והעברה של אותות ופודוט מוזיקליות בין כלים נגינה אלקטרוניים (דוגמת סינטיסיזר) לרבות כלים נגינה וירטואליים המנוהלים בתוכנת מחשב. פודוט מוזיקליות אלו מכילות הוראות לכלים הנגינה האלקטרוניים (והוירטואליים) על איזה צלילים עליהם לנגן ואיזה צלילים להפסיק לנגן, באיזו עצמה יש לנגן כל צליל, מהו כל הנגינה הוירטואלי שאחראי לבצע את הניגון ועוד. בטרמינולוגיה של MIDI, הודעות ופודוט אלו קרויים אירועים (Events). על-כן נשמש ב莫斯ג זה בהמשך במקומות הודעות/פודוט/אותות.
- **קובצי MIDI:** הפורמט הנפוץ להעברת אירועי MIDI בתוכנות מחשב הוא קובץ MIDI סטנדרטי עם סיומת .mid, למשל "mySong.mid". קבצים אלו מוגדרים בסטנדרט SMF (Standard MIDI File), והם פופולריים מאוד בשימוש בתוכנות עריכה של מוסיקה, לאחר שהבדיל מפורטים נפוצים אחרים של קבצי שמע כגון mp3 או wav שמכילים ממש audio, קבצי המIDI אינם מכילים audio כלל, אלא רק תיאורי מוסיקלי שלו ע"י אוסף אירועי MIDI הכוונים פודוט שעלה כל הנגינה האלקטרוניים והוירטואליים לפענה ולבצע בשליל להפיק את הנגינה המתבקשת.
- **יתרונות וחסרונות המIDI:** שיטה זו של קידוד האירועים במקומות הקלטה ה-audio עצמה אמנים אינה מושלמת, וכוללת חסרונות מסוימים – כפי של.setDefaultPageNum יש חופש מסויים באופן הניתוח, פענוח והציגה של דפי HTML, כך גם לתוכנות שמעם יש מרוחק מסויים של חופש בשימוש הסטנדרט, כך שתוכנות שונות עלולות לענח את אירועי המIDI טיפ טיפה אחרת אלו מallow במובנים של אפקטים של כל הנגינה, קובוצת צלילים של קבוצות צלילים ועוד, כך שתוצר השמעה עשוי להיות לאחר מכן מתרגן ע"י תוכנות שמעם שונות. חישרונו נוסף הוא שהסאונד הסופי הוא מלאכוטי (וירטואלי), ולכן לא יהיה הטכנולוגיהאפשרה לתחקורת אובי הניגון האנושי באופן די מדויק, הביצוע של הניגון המלאכוטי לא מכיל את כל הדקויות של הניגון האנושי כך שההצורה הסופית לא בהכרח משקף את כוונת המשורר (או ליתר דיוק – כוונת המלחין/מחבר הקובץ). עם זאת, לקובצי המIDI יתרונות משמעותיים מאוד –
 - א. חישרונו: בעוד שקובצי שמע שמכילים audio כגון mp3 ו-wav מגיעים בקלות למשקל שגע בין כמה MB בודדים לעשרות ואף מאות KB, קובצי MIDI טיפוסיים יותר ריק כמה עשרות או מאות בודדות של KB, שכן קל לאחסן ולהזיר אותם ולהעבירם בראש על פני ערוץ תקשורת בין שרת ללקוח ובין מחשב לכל נגינה אלקטטרוני.
 - ב. הקבצים "פתוחים" או "חסופים" ב__[מונח]__ מוכאים רשיימה של כל האירועים המוזיקליים שעל כל הנגינה האלקטרונית והוירטואלית להגיב אליהם על מנת לנגן את המוסיקה המתוארת בקובץ. לפיכך, ניתן להחלץ את ה-"data" מתוך הקובץ, לעורך אותו, ולשמור אותו בחזרה. קיימות מגוון תוכנות בשוק, לרבות תוכנות בקוד-פתוח, התומכות בייבוא ויצוא של קובצי MIDI, שמספקות שירות של קריית תוכוםם של מיצגים, שינוי גובה הסולם של האירועים, הרמת המIDI לקובץ audio, ועוד. למעשה, קובצי המIDI כל כך פופולריים בתעשייה היום שככל תוכנה בתחום עריכה וניגון של תוכום שמכבדת את עצמה מוכרכת לספק תמיכה בקרייה וכ כתיבה של תוכן "מ-" / "אל" קבצי MIDI.

- **מבנה כללי של קובצי המIDI:** קובצי המIDI מחולקים לבלוקים (Chunks) כאשר אחד מהם הוא בלוק כותרת (Header Chunk) הכלול אironui metadata על הקובץ, דוגמת תיאור טקסטואלי של שם הקטע המוסיקלי, מאפייני מקצב וטמפו (BPM) ועוד, ואילו יתר הבלוקים מכילים אironuis רצועות (Track Chunks). לרוב כל רצעה מכילה אironuis המיועדים לכל נגינה יחיד. בכך למשז את, המIDI מנהל ערוצים (channels), כך שכאשר נשלח אironui לערוץ מסוים, רק רכיבים שמאזינים לאironuis על אותו ערוץ מגיבים לאironuis אלו. אפשר להסתכל על הערוץ כמנוע אאנולוגי למושג ה-port ב-socket-ים בתקשורת. בהקשר של תוכנת שמע, הרכיבים הם כל נגינה וירטואליים שונים, למשל על ערוץ 1 מאזין פסנתר, ערוץ 2 גיטרה, וכן הלאה. הסטנדרט תומך بعد 16 ערוצים שונים. קיימים גם פורמטים מיידי המכילים את כל האironuis על רצעה אחת ויחידה (Format 0). האופי המונוליטי של פורמט זה מסבך מאוד את ניתוחו וערכיה של אironuis אינדיווידואליים, על כן יישום זה תומך רק בפורמט מיידי שותומך בריבוי רצועות (Formats 1-2).
- **קובצי מIDI בקובצי Shmu:** אמנים קובצי מIDI כאמור אינם מכילים אודיו, אלא רק אוסף אironuis, אך לאור הפופולריות הרבה של ה-MIDI, מרבית יישומי נגינה המוסיקלית ב-PC וממשיריהם סוללארים (Media Player, VLC, Winamp ועוד) תומכים בקובצי מIDI, וידועים לפענה אותם ולגנן את אironui המIDI שבקבצים אלו על כל נגינה וירטואליים.

5.2.2 ארכיטקטורת תת-שכבות שירותים ה-MIDI

להבדיל מפלטפורמות אחרות (למשל Java) שספקות ספריות סטנדרטיות מובנות לתמייה בפרוטוקול MIDI, פלטפורמת .NET. לא מספקת ספריות שכאה. עם זאת, ישן מספר ספריות ופרויקטים בקוד-פתוח שנិנו לעשות בהן שימוש, חלקן מיעדות לתוכנות מול כל נגינה אלקטרוניים כמו סינטיסיזר וספקות רק תמייה בעבודה ישירה מול אironui המIDI ע"פ הפרוטוקול עצמו (API) ברמת low-level שדורש היכרות מעמיקה עם פרטיה הפרוטוקול, ואחרות עוטפות את הפרטימ השכניים של הפרוטוקול וספקות משק API ברמת High-Level שמבצע אבסטרקציה על הפרוטוקול ומאפשר לקליננטים לדבר בשפה של יישויות מוסיקליות כמו תווים ולא בשפה של אironui מIDI.

מערכת Soloist עשויה שימוש בספרייה הבאה, שמאפשרת עבודה מול מIDI גם ברמת האironuis ב-
low-level וגם ברמת הפשטה גבוהה יותר high-level בהתאם לצורך –

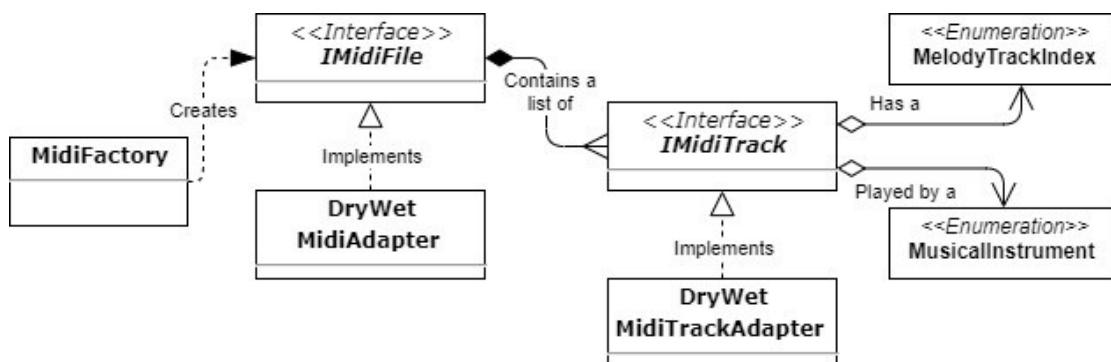
<https://github.com/melanchall/drywetmidi>

תת-שכבות שירותים ה-MIDI מסתירה את פרטיה המימוש משאר תתי-השכבות והשכבות האחרות במערכת, ובפרט אינה חושפת את שירותים הספרייה החיצונית שלעליל, אלא מכמיסה אותה ועוטפת אותה במחלקת Adapter שומרה את השירותים הנדרשים ממנה לשירותים שיתר השכבות מכירות במנשך IMidiFile שיפורט לעיל, כך שאם בעתיד יוחלט להחליף את מימוש שירות ה-MIDI בספרייה אחרת (למשל אם Adapter Microsoft יפתחה עתיד ספרייה ל-Interface, ולאחרן את הזרקת התלות (Dependency Injection) של הקליננטים עם ה-Adapter המעודכן).

להלן רכיבי הפיתוח המוגדרים בתת-שכבה זו –

#	תפקיד	שם הרכיב	סוג רכיב פיתוח
1	מספר רצועת המנגינה בקובץ ה-MIDI	MelodyTrackIndex	Enumeration
2	ייצוג כל נגינה וירטואליים ב-MIDI	MusicalInstrument	Enumeration
3	רצועת כותרת/נגינה בקובץ MIDI	IMidiTrack	Interface
4		DryWetMidiTrackAdapter	Class
5	MIDI קובץ	IMidiFile	Interface
6		DryWetMidiAdapter	Class
7	יצירת קובצי MIDI	MidiFactory	Class

להלן דיאגרמת מחלקה המתארת את הקשרים שבין רכבי הפיתוח המרכזיים במודול זה –



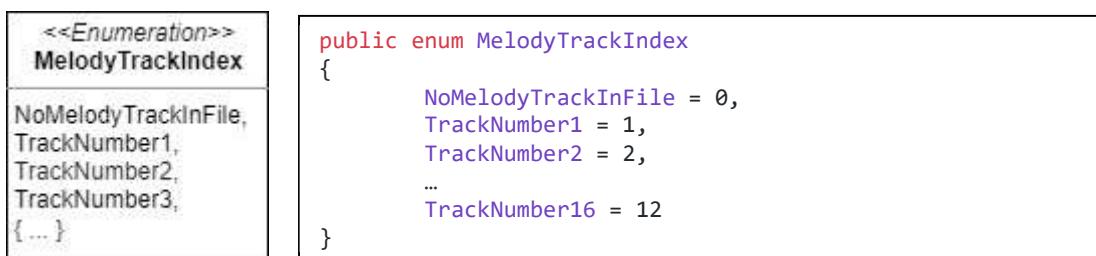
5.2.3 אוסף מספרי רצועות (MelodyTrackIndex Enumeration)

כאמור, אחד היתרונות הגדולים בקובצי MIDI הוא שניתן לקרוא את תיאור האירועים המוסיקליים, לעבד אותם תוך הפעלת לוגיקה עסקית ולבסוף לכטוב בחרזה את האירועים המעודכנים וע"י כך להפיק קובץ MIDI חדש/מעודכן שניתן להשמיון בנגני מוזיקה.

בהקשר של אפליקציה זו של הלחנת מנגינות חדשות על בסיס יצירות קיימות, האפליקציה צריכה לספק יכולת להחליף מנגינה קיימת. בקובצי MIDI ניתן לבצע בזאת בклות ע"י החלפת הרצועה שמכילה את אירועי המנגינה המקורייה בrzועה חדשה שמכילה את אירועי המנגינה המעודכנת. בכך שהאפליקציה "תדע" איזו רצועה יש להחליף (זהינו איזו רצועה מכילה את המנגינה המקורייה), על המשמש לספק את מספירה הסידורי. האפליקציה העצמה אינה יכולה להסיק נתון זה בעצמה שכן בקובץ MIDI טיפוסי ישנו מספר רצועות המכילים אירועים של ניגון שונים. אולם, ניתן לבצע איזשהו פילוח והשווואה בין הרצועות ולבדוק איזו מהרצועות מכילה אירועים, ככלותם היוצרים מנגינת יחיד, או אם מקרים מורכבים במיוחד שבו המנגינה חזרה על עצמה בכמה רצועות שונות בקבולות שונות (למשל במרוחות של טרצה), על-כן בכל מקרה החלטה זו של איזו מהמנגינות יש להחליף צריכה להגיע מהמשתמש.

פרוטוקול MIDI תומך بعد 16 ערוצים, ובדי'כ', לפחות במסיניקה קלה, להקה או הרכב כוללים רק כמה נגנים בודדים (פחות מ-16) כך שכל תפקיד (כלי נגינה) כך שניתן להקצתו לכל כלי נגינה/תפקיד רצועה נפרדת עם ערך שידור מסוימת. נקודת ההנחה היא איפה שקובץ MIDI יכיל 16 רצועות לכל היוטר (בהתאם למוגבלות 16 הערוצים). בכך לאכוף אילוץ זה בклות תוך הגדרה בעלת משמעות ברורה, הוגדר בקובץ MelodyTrackIndex.cs אוסף קבועים עבור אינדקסים של 16 רצועות אפשריות בקובץ, כאשר הספרה מתחילה ב-1 (one-based indexing). כמו כן, הוגדר קבוע אחד לייצוג המקרים שבהם כוונת MIDI היא פלייבק "טהור" שמכיל רק את הלויוי ללא מנגינה, כלומר המקרים שבהם נדרש לשבץ מנגינה בקובץ רצועה חדשה מבלי להחליף רצועה קיימת.

להלן הגדה הבסיסית של אוסף הקבועים –



אוסף זה שימושי להעברת הפורט של מספר רצועה בקובץ MIDI בין שכבות האפליקציה השונות כישות מוגדרת היטב (strongly typed), ומונע שימוש ב-"ערכי קסט קבועים".

5.2.4 אוסף כלי נגינה (MusicalInstrument Enumeration)

ב-MIDI מוגדרת רשימת קודים כלליים ([General MIDI Program Change Instruments](#)) המיצגים 128 כלי נגינה ו/או אפקטים שונים, המוחלקים למשפחות – פסנתרים, כלי נשיפה, כלי מיתר, כלי קשת ועוד.

עדי שימוש בערכים אלו באירועים המתאים ב프וטוקול ה-MIDI, ניתן להגדיר איזה כלי נגינה/אפקט יבצע אירוע נתון, מה שמאפשר את היכולת לתת למשתמשי הקצה את האפשרות לבחור את כלי הנגינה המבוקש לביצוע הנגינה החדש. בכך ניתן שימוש ב-"ערכי כסם" קבועים שמייצגים את כל הנגינה והאפקטים השונים, הרשימה הוטמעה במערכת Soloist כאוסף קבועים ([DryWetMidi](#)) בקובץ ([Enumeration](#)). אמנים רשימה זו כבר מוגדרת בספרייה החיצונית [DryWetMidi](#) שהמערכת עשויה בה שימוש, אולם לאור הרצון לתיחס ולכמוש את כל פרטי הספריות החיצונית מיותר שכבות המערכת, הוחלט להגדיר רשימה זו מחדש כך שלא תהיה תלולה בספרייה החיצונית שבה נעשו שימוש לאספקת שירות ה-MIDI. כמו כן, הרשימה שבנה שמשה במערכת היא רשימה חלקית בלבד: היא מדירה רק את 112 הערכים הראשונים ואינה תומכת (במכוון) ביותר 16 הערכים של משפחות כלי ההקשה אפקטים, שאינם רלוונטיים לניגון מנגינות.

בדומה לאוסף הקבועים של [MeodyTrackIndex](#), גם אוסף זה שימושי להעברת הפרמטר (של כל הנגינה המבוקש) בין שכבות האפליקציה השונות כישות מוגדרת היטב ([strongly typed](#)), ומונע שימוש ב-"ערכי כסם קבועים".

להלן ההגדרה הבסיסית של אוסף הקבועים של כל הנגינה –

<<Enumeration>>	MusicalInstrument
AcousticGrandPiano, BrightAcousticPiano, ElectricGrandPiano, { ... }	<pre>public enum MusicalInstrument { AcousticGrandPiano, BrightAcousticPiano, ElectricGrandPiano, ... }</pre>

5.2.5 ממשק רצועת MIDI (IMidiTrack Interface)

המנשך IMidiTrack מייצג רצועה מוסיקלית בודדת בקובץ MIDI. הוא מתפרק מבנה בלבד (POCO – Plain Old CLR Object) ללא פונקציונליות, המכיל מקבץ של מאפיינים רלוונטיים ו-"משמעותיים" של רצועה מוסיקלית בקובץ MIDI: מספירה הסידורי של הרצועה בקובץ MIDI (אינדקס), שם הרצועה, קוד כל הנגינה הוורטואלי שמבצע את אירוע ה-MIDI של רצועה זו, ושם כל הנגינה הוורטואלי המתאים לקוד כל הנגינה שלעיל –

Property	Type	Description
TrackNumber	MelodyTrackIndex (Enumeration)	One-based ordinal number of the track in its containing midi file.
TrackName	string	Midi sequence track name.
InstrumentMidiCode	MusicalInstrument (Enumeration)	The general MIDI program number which identifies this track's MIDI musical instrument.
InstrumentName	string	The instrument description that corresponds to the instrument code.

5.2.6 ממשק קובץ MIDI (IMidiFile Interface)

המנשך IMidiFile מייצג קובץ MIDI סטנדרטי (SMF) בעל הסיומת .mid. ממשק זה הוא ממשק "lightweight", במובן שהוא אינו מכיל את כל התוכנות והfonctionnalities שהיינו מצפים שייהו במנשך המקורי ישות של קובץ MIDI, אלא רק את מה שהכרחי לישום המערכת. האינטראקציה המלאה מול קבצי MIDI מבוסעת כאמור באמצעות שירותים של הספרייה החיצונית [DryWetMidi](#), אולם הגישה אליה היא רק באמצעות מחלקה Adapter ([DryWetMidi](#) פירוט בהמשך) שimplements את הממשק IMidiFile Interface שאותו יתר השכבות מכירות ודרכו הן צורכות את שירותים ה-[MIDI](#) של תחת-שכבה זו.

להלן המאפיינים (Properties) המוגדרים במנשך זה –

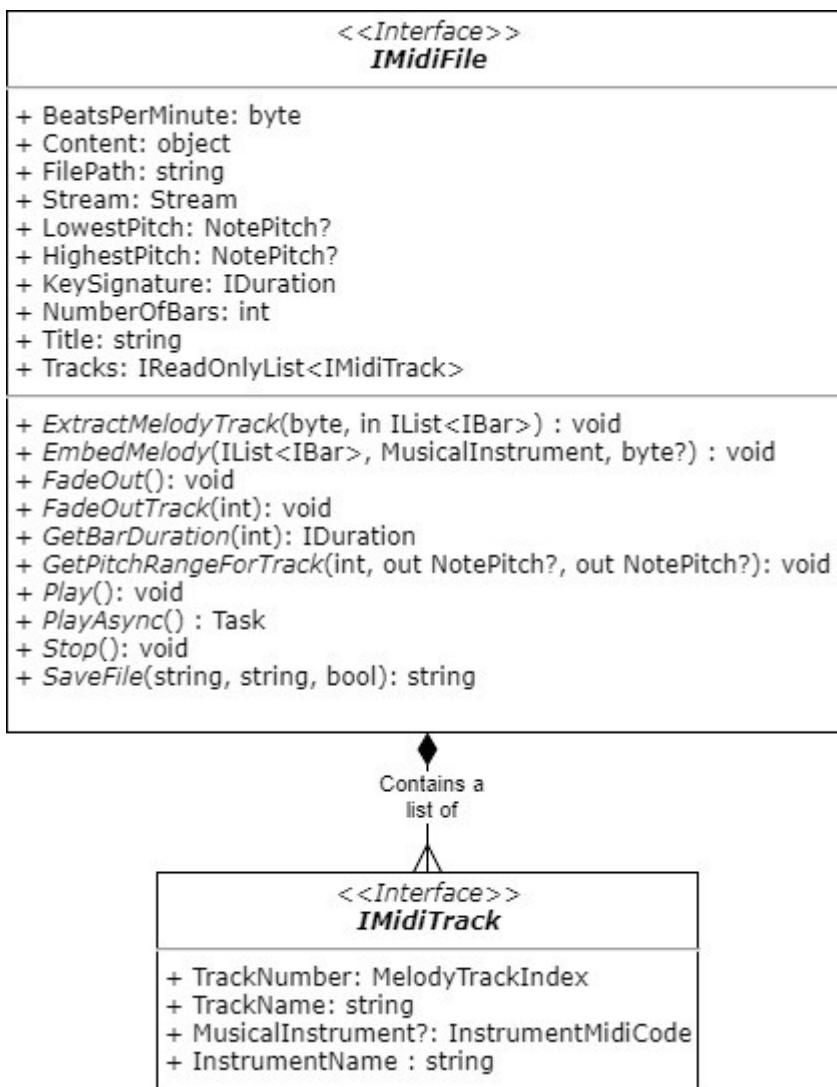
Property	Type	Description
FilePath	string	Absolute physical path of the MIDI file.
Stream	Stream	Input source stream which is used to read the midi content from.
Content	object	The actual content the MIDI file.
Title	string	MIDI Sequence Name from the header track meta events.
BeatsPerMinute	byte	BPM – Beats Per Minute. The tempo (speed) that is set for playing the events in the MIDI file.
NumberOfBars	int	Total number of bars in the MIDI file.
KeySignature	IDuration	The key signature from the MIDI meta events.
LowestPitch	NotePitch?	Global lowest pitch in the entire midi file.
HighestPitch	NotePitch?	Global highest pitch in the entire midi file.
Tracks	IReadOnlyList <IMidiTrack>	The tracks (chunks) contained in the MIDI file.

כל המאפיינים במנשך זה הם לקריאה בלבד. המערכת אינה תומכת בשלב זה בעדכון תוכן קובץ MIDI [למעט עדכון הטמעת מנגינה החדשה בסיום תהליך הלחנה], אולם עדכון זה נעשה באמצעות מתודות [יעודיות](#) ([EmbedMelody](#)- ו-[ExtractMelodyTrack](#)) , ולא ע"י מניפולציה על מאפיין הרצויות (Tracks) שמוגן מפני עדכון].

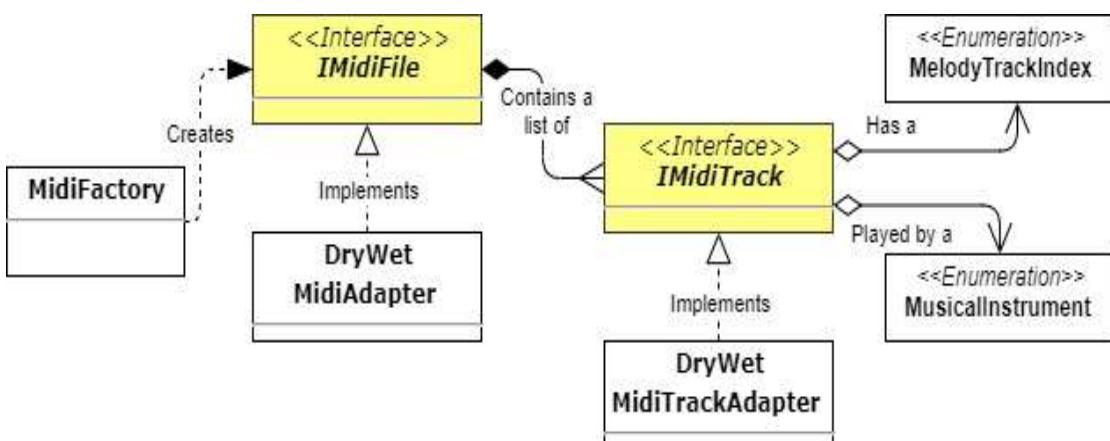
להלן תקציר השירותים (Methods) המוגדרות במנשך זה –

Method	Description
EmbedMelody	Converts a melody contained in a collection of bars into a midi track and adds it to the midi file.
ExtractMelody Track	Removes the requested track from the midi file, and returns the removed track content as a bar sequence in a music-theory representation, containing the notes from the extracted melody track.
FadeOut	Fades out the volume of all played notes towards the end of the melody on all tracks / on a single specified track.
GetBarDuration	Returns the specified bar's duration.
GetPitchRange ForTrack	Returns lowest and highest pitches contained in the given midi track (the return is via out parameters).
Play	Starts playing the MIDI events contained in the file synchronously (blocking interactive input during playback).
PlayAsync	Starts playing the MIDI events contained in the file asynchronously (non-blocking playback).
SaveFile	Saves midi file on local device.
Stop	Stops playing the MIDI data.

להלן דיאגרמת מחלקה עבור המנשכים של רצועת MIDI וקובץ MIDI



להלן דיאגרמה המדגישה את הקשר שלhn ליתר רכיבי הפיתוח במודול ה-MIDI



5.2.7 מחלקה מעטפת לקובץ MIDI (DryWetMidiAdapter Class)

המחלקה DryWetMidiAdapter מimplements את הממשק [IMidiFile](#), תוך שימוש בספרייה עוזר חיצונית – [DryWetMidi](#), שאורה היא עוטפת ומסתירה (מכמיסה) וחושפת כלפי חוץ את הממשק הפשט [IMidiFile](#) שלו מצפים הקליניטים, ומכאן שמו המחלקה (על שם דפוס העיצוב Adapter.).

הספרייה החיצונית אמונה מספקת הפעלה לפרוטוקול ה-MIDI, אך עדין מכילה API יחסית "Low-Level" שדורש היכרות עם שמות אירועים ב-MIDI שבהם אחוור ועדכו נתונים בקובץ. המחלקה אחראית על התעסקות עם ה-"עובדת שחורה" זו של אינטראקציה מול ה-API של הספרייה החיצונית, ולספק לקליניטים יותר רכיבי הפיתוח בשכבה זו ובשכבות אחרות במערכת) פשטוט, נקי וקל לתפעול.

משתנים: המחלקה מגדרה את סט המשתנים הפרטיים הבאים –

Field	Type	Description
_midiContent;	MidiFile	Delegate to DryWetMidi library midi file entity.
_tempoMap;	TempoMap	DryWetMidi library property for managing the timespans & tempo in the midi file.
_metadataTrack	TrackChunk	Header chunk of the midi file.
_metaEvents	List<MidiEvent>	MIDI events from the header chunk.
DefaulTrackName	string	Default name that would be used for the composed melody track chunk in the midi file.
_midiPlayer;	Playback	Medium for playing the MIDI files events on an output device.

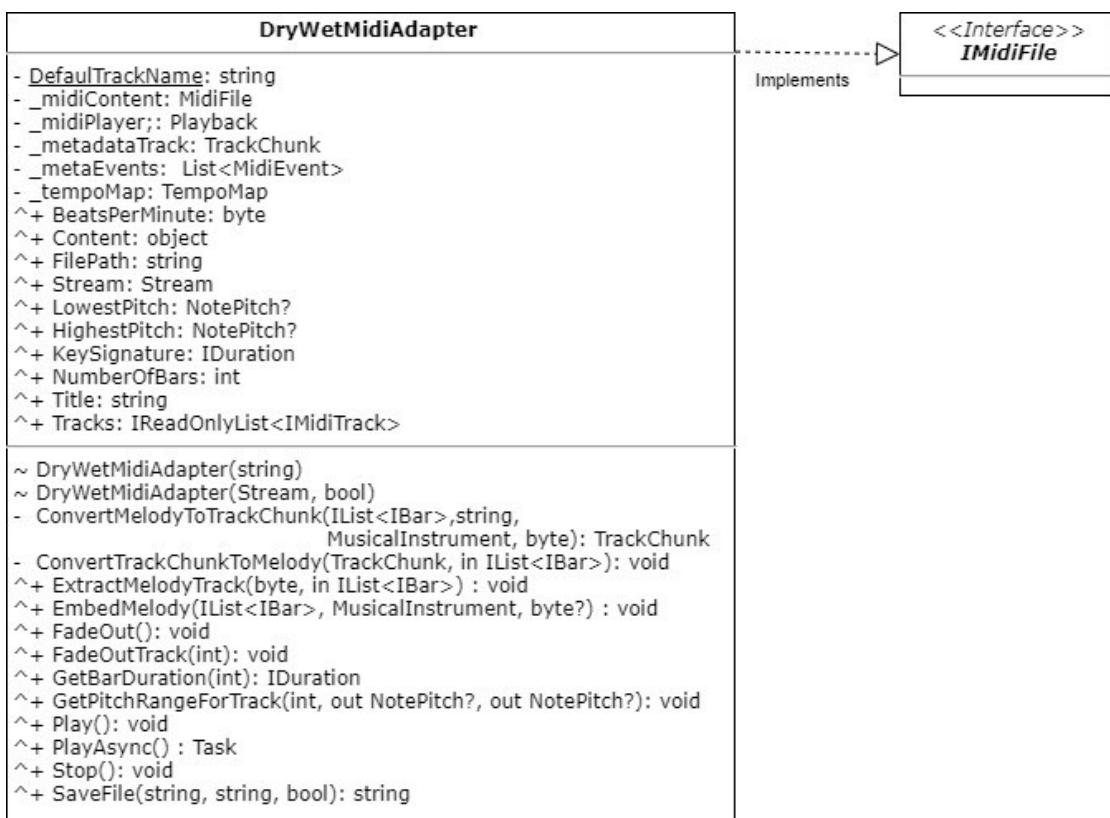
בנייה: המחלקה תומכת בנייןaternals – האחד מקבל כקלט נתיב לקובץmidi והשני מקבל זרם בתים של הקובץ (שימושי בעיבוד תוכן קובץ שכבר נקרא ונמצא בזיכרון התכני) –

Signature	Description
DryWetMidiAdapter(string)	Construction based on given path to the actual MIDI file.
DryWetMidiAdapter(Stream, bool)	Constructs an instance based on a given stream of the actual MIDI file content and a boolean flag of whether to close the given stream when it's fully read.

מתודות – מעבר למאפיינים והמתודות במנשך [IMidiFile](#) שמומומשות במחלקה זו, מחלקה זו מגדרה גם שתי מתודות פרטיות ייעודיות שאחרואיות על אדפטציה מול הספרייה החיצונית [DryWetMidi](#) ע"י המרה בין הייצוג פנימי של ישוויות המערכת אל ייצוג הספרייה החיצונית ולהפוך: מתודה אחת ([ConvertMelodyToTrackChunk](#)) לאחריות להמיר ייצוג של מנגינה המורכבת מישויות התאורה המוסיקליות שתוארו בפרק [יצוג ישויות מוסיקליות](#) אל ישות חיצונית המייצגת רצואה בקובץ MIDI בספרייה החיצונית, ומתודה שנייה שמיושם בשירותי הספרייה החיצונית: אלגוריתמי הלחנה עובדים על הישויות המוסיקליות, ומפיקים בסיום תהליך הלחנה מנגינה, שהיא פשוט אוסף של תיבות (ישויות [IBar](#)). בכך לשבץ תיבות אלו בקובץ MIDI יש להמיר אותם לספרייה החיצונית שמספקת את שירותי ה-MIDI השונים. באופן אנלוגי, לאחר קריאת קובץ MIDI, יש להמיר אותה מהייצוג בספרייה החיצונית כרצואה לסדרת תווים בתיבה ע"י הייצוג הפנימי במערכת. להלן תקציר המתודות –

Method Name	Description
ConvertMelodyToTrackChunk	Converts a melody encoded as a note sequence in a list of bars in internal representation to a MIDI track chunk in external third-library (DryWetMidi) representation.
ConvertTrackChunkToMelody	Converts a midi track from a midi file represented in in external third-library (DryWetMidi) representation into a collection of musical notes in internal representation.

להלן דיאגרמת מחלקה עבורה – DryWetMidiAdapter



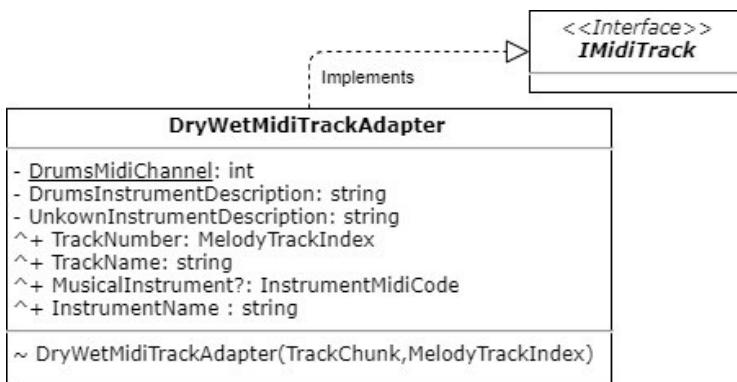
5.2.8 מחלקה מעטפת לרצועת MIDI (DryWetMidiTrackAdapter Class)

מחלקה זו (DryWetMidiTrackAdapter) מimplements את הממשק [IMidiTrack](#). בדומה למחלקה [DryWetMidi](#) היא עוטפת ומסתירה (מכמישה), כך שכלפי חוץ נחשף רק הממשק הפשט [IMidiFile](#) שהקליניטים מצפים לו, ומכאן שם המחלקה (Adapter). יתר על כן, לאחר שזויה אוטי ספריית>User Chironight שמשמשת את המחלקה [DryWetMidiAdapter](#), ומאהר שהבנאי של ייצוג בפרמטר מהספרייה החיצונית שאינו אמור להיות מחוץ למחלקה, המחלקה של ייצוג רצועה הוגדרה כמחלקה פנימית פרטיטית (inner class) המקוונת בתוך המחלקה המייצגת קובץ. זה גם מתиישב עם העבודה שרצועה רלוונטיות רק תחת קונטקט של איזשהו קובץ, כך שambilial קליניטים מחוץ למחלקה לא אמורים לקבל גישה ישירה לרצועה כישות עצמאית, אלא כישות המורכבת בתוך ישות של קובץ, ככלומר הגישה לרצועות היא מתוך הקובץ המכיל אותן.

המחלקה אינה מדירה פונקציונאליות נוספת מעבר לזה של הממשק [IMidiTrack](#) כזכור, ממשק זה מספק רק מבנה (POCO) עם סט מאפיינים לקריאה בלבד, כך שככל מהות מחלקה זו היא לספקIMPLEMENTATION לאחזר מאפיינים אלו תוך שימוש ב-API של הספרייה החיצונית.

בנייה: המחלקה תומכת בבניין יחיד המקבל שני פרמטרים – פרמטר ראשון הוא רצועת MIDI [TrackChunk](#) שמודדר בספרייה החיצונית [DryWetMidi](#), ופרמטר שני הוא מספירה הסידורי של הרצועה בתוך קובץ ה-MIDI שמכיל אותה. כאמור, הפרמטר הראשון הוא טיפוס מהספרייה החיצונית שאינו אמור להיות מוכרך מחוץ למחלקה, אולם לאחר שזויה מחלקה פרטיטית מקוונת, הגישה לבניין נעשית רק מתוך המחלקה המכילה שambilial עשויה שימוש בעצמה בספרייה החיצונית.

להלן דיאגרמת המחלקה –



5.2.9 מחלקה מפעל לישויות MIDI (MidiFactory Class)

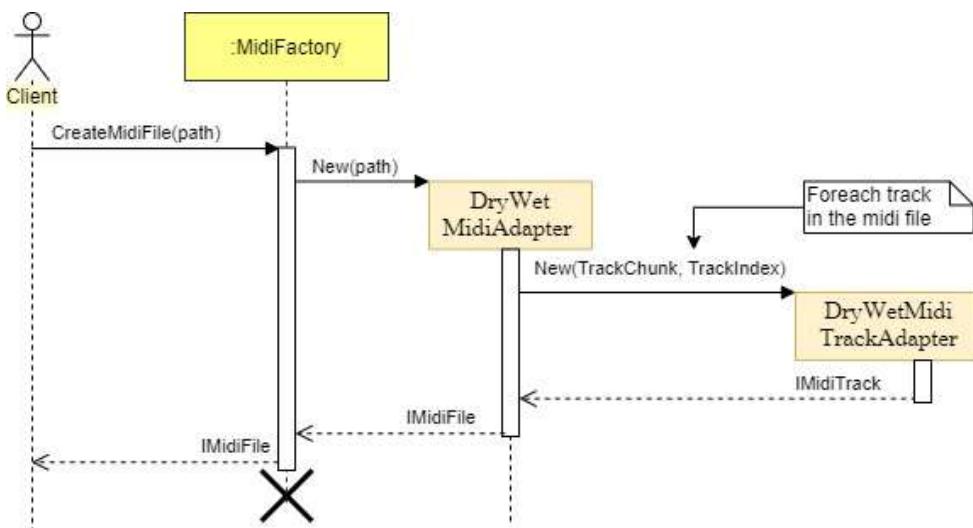
מחלקה זו היא מחלקה שירות סטטית המספקת מתודות שירות לצירוף מופעים קונקרטיים שemmמשים את הממשק [IMidiFile](#), ומכאן שם המחלקה (על שם דפוס העיצוב **FactoryMethod**). היא כוללת שני מתודות סטטיות לצירוף מופעים –

```

internal static IMidiFile CreateMidiFile(string midiFilePath) {...}
internal static IMidiFile CreateMidiFile(Stream stream, bool disposeStream = false) {...}
  
```

מוטיבציה מחלקה זו היא לשבור את התלות של קליניטים בIMPLEMENTATION של שירות MIDI, ולהשאיר תלות רק באבstarטרכיה, זהינו רק במנשך עצמו. קליניטים המונינים בשימוש בשירותי MIDI יפנו למחלקה **MidiFactory**-ה. בבקשת לקבלת מופע של המنشך, והמחלקה תהיה אחראית לספק מופע שכזה, וכל זאת בעלי שהקליניטים צריכים להכיר את המחלקות שמספקות את המימוש של המنشך בפועל מאחורי הקלעים.

נכון לעכשו, המימוש של ה-Factory כולל פשוט יצירה מופעים של מחלקה ה-[DryWetMidiAdapter](#), שכן זהו המימוש היחיד המוגדר בעת המערכת. אם באותו יהיה צורך בהחלפת המימוש של שירות ה-MIDI, מלבד המימוש עצמו של המنشך [IMidiFile](#) (ו-[IMidiTrack](#)) במידת הצורך) כל שיידרש הוא לבצע את ההחלפה במימוש הפנימי של מתודות ה-Factory, לקליניטים עצם ה החלפה תהיה "סקופה", הם ימשיכו לפנותו לריגיל אל ה-Factory, ככלומר, לקליניטים עצם ה החלפה תהיה "סקופה", הם ימשיכו לפנותו לריגיל אל ה-Factory, בבקשת לקבלת ספק שירות MIDI ויקבלו מופע של המنشך שמספק שירות זה, מבלי בכלל לדעת מהי המחלקה הקונקרטית שמספקת את המימוש. להלן דיאגרמת רצף הממחישה כיצד מבקשים ומקבלים מופע של [IMidiFile](#) שכבר עוטף בתוכו אוסף רצוות מטיבוס [IMidiTrack](#)



5.3 הלחנת מנגינות (Composition Service)

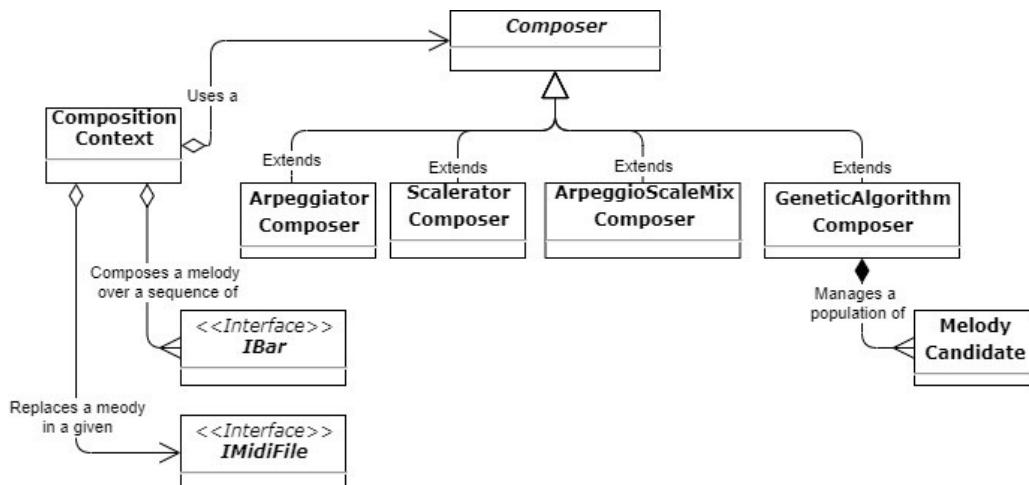
תת-שכבה זו של הלחנת מנגינות מחולקת למספר מרחבי שמות (Namespaces) כלהלן –

Namespace	Usage
CW.Soloist.CompositionService	Provides clients a public high-level interface for using the various composition algorithms interchangeably, abstracting way low-level details.
CW.Soloist.CompositionService.Composers	Provides an abstract skeleton and infrastructure for all composition algorithms.
CW.Soloist.CompositionService.Composers.*	Various concrete composition algorithms which implement the abstract derived operations.

להלן רכיבי הפיתוח המוגדרים בתת-שכבה זו –

#	פקוד	סוג רכיב פיתוח	שם הרכיב
1	מחלקה קונטקט הלחנה	Class	CompositionContext
2	מחלקה מלchein אבסטרקט	Class	Composer
3	מחלקה מלchein פירוקי אקורדים	Class	ArpeggiatorComposer
4	מחלקה מלchein סולמות	Class	ScalculatorComposer
5	מחלקה מלchein סולמות ופירוקי אקורדים	Class	ArpeggioScaleMixComposer
6	מחלקה מלchein עם אלגוריתם גנטי	Class	GeneticAlgorithmComposer
7	מחלקה מנגינה מועמדת באלגוריתם	Class	MelodyCandidate
8		Class	
9	אוסף קבועים לייצוג אלגוריתמי הלחנה	Enumeration	CompositionStrategy
10	מבצע יצירה מלchein ע"פ אלגוריתם הלחנה	Class	ComposerFactory
11	מחלקה לשירותי הלחנה	Class	ComposerExtensions
12	מחלקה ל-Enums	Class	EnumExtensions
13	קבוע ערך שונה לסיווג פרמטרים המועברים למתחות השונות להלחנה.	Enumeration	ChordNoteMappingSource
14			DurationSplitRatio
15			NoteSequenceMode
16			OverallNoteDurationFeel
17			Permutation
18			PitchRangeSource
19			SortOrder

להלן דיאגרמת מחלקה המציג את היחסיות המרכזיות במודול זה והקשרים שביניהם –



5.3.1 מחלקה קונטקט הלחנה (CompositionContext Class)

המחלקה CompositionContext משמשת כרכיב ה-context בדף העיצוב Strategy Pattern אשר מוגדרת כמחלקה של מנגינות, ומכאן שם המחלקה.

מחלקה זו אחראית על ניהול האינטראקציה ואינטגרציה עם כל הרכיבים השונים בשכבה הולוגית העסקית, ונמצאת בקו החזית של השכבה מול הקליענים: שכבות אחירות המוניות בשירותי הלחנה פונים אך ורק למחלקה ה-CompositionContext והיא זו שஅחראית על עיבוד הבקשה שלהם, בדיקת תקינות הקלט, ובמידה והבקשה תקינה, מחלקה זו אחראית להעיבר אותה להלאה אל הגורמים הרלוונטיים להמשך טיפול – ראשית היא מעבירה את הבקשה אל אלגוריתם הלחנה לחיבור מנגינה, ולאחר מכן עם קבלת מנגינה מאלגוריתם הלחנה המחלקה פונה לתת-שכבה שירות-IDI-MIDI לשיבור מנגינה שהוחננה בקובץ MIDI שאוטו לבסוף היא מחזירה כפלט לקליענים.

באופן זה, הקליענים לא צריכים להיות מעוררים ב-low-level של פרטיה השימוש של מה מבוצע ע"י אלגוריתם הלחנה ומה מבוצע ב-MIDI. הם יכולים פשוט לפנות למחלקה קונטקט הלחנה בבקשת מנגינה וזו תdag למל את כל האופציות מהורי הקלעים ולספק להם את הפלט המבוקש תוך הסתרת כל הפרטים הטכניים. מחלקה זו משתמש גם כمعין "Facade" של שכבה הולוגית העסקית.

יתר על כן, שימוש בדף העיצוב Strategy Pattern עבור אלגוריתם הלחנה מאפשר לקליענים לבחור את אלגוריתם הלחנה שבו יישא שימוש לחיבור מנגינה, וכל זאת בזמן ריצה ללא צורך בהידור חדש של התכנית. בכך לעציוו המימוש העיקרי העיקרי מבין אלגוריטמי הלחנה הוא אלגוריתם גנטי, שהינו אלגוריתם הבוסס על שיטות חיפוש היוריסטיות למציאת פתרונות אופטימליים (במקרה הזה – מנגינות שנשמעות היטב), יתר פירוט על אלגוריתם זה בהמשך בפרק [הלחנה עם אלגוריתם גנטי](#). לאחר שזהו האלגוריתם העיקרי, הוא נבחר כברירת מחדל אם הקליינט לא ציין בחירה אחרת. מלבד אלגוריתם זה, הוגדרו מספר אלגוריתמי הלחנה בסיסיים ביותר שמהוללים רצף תווים על בסיס מהלך האקורדים, הסולמות המומופים לאקורדים, או שילוב ביניהם. אלו כאמור אלגוריתמים מאוד בסיסיים ולמעשה משמשים כאביוני בניין באלגוריתם הגנטי המרכזי. הם הוגדרו כאלגוריתמים עצמאיים לבדיקה והמחשה של אפשרות השימוש בדף העיצוב של האסטרטגיה, ככלומר לבדיקה והמחשה של היכולת לשנות את אלגוריתם הלחנה המבוקש בזמן ריצה. הבחירה היא בכל אופן לעשות שימוש באלגוריתם המרכזי – האלגוריתם הגנטי, וכן, מערכת-Web חושפת כת רק את האלגוריתם הגנטי, אך ניתן לשנות זאת בקלות ביחסים-Web ו/או במערכת האבטיפוס של מימוש Windows Forms .Console

5.3.1.1 בנאים (Constructors)

המחלקה מגדרה שני בנאים, המקבלים את פרטי הקטע המוסיקלי שעבורו יש להלחין את המנגינה החדשה. בגין אחד מצפה לקבל מופעים "חיים" בזיכרון של תוכן קובץIDI-MIDI ומhalt האקורדים, ואילו הבנאי השני מספק תמיכה של קריאת תכנים אלו מתוך קבצים ומצפה לקבל נתיבים לקבצים אלו. בפועל הבנאי השני קורא את התוכן מהקבצים, יוצר מהתוכן מופעים של האובייקטים המתאימים ומפנה אותם להמשך טיפול אצל הבנאי הראשון. הבנאים אחראים על תחילת מופע הקונטקט עם פרטי הקטע המוסיקלי תוך בדיקות תקינות נתונים של תוכןIDI-MIDI, האקורדים, התאמות ביןיהם וההתאמות בין קובץIDI-MIDI לאינדקס הרצועה שמצוינת את מספר רצועת המנגינה המקורי שיש להחליף בקובץ. להלן חתימות הבנאים –

```
public CompositionContext(IList<IBar> chordProgression, IMidiFile midiFile,
    MelodyTrackIndex? melodyTrackIndex = null) {...}

public CompositionContext(string chordProgressionFilePath, string midiFilePath,
    MelodyTrackIndex? melodyTrackIndex = null) {...}
```

תכונות המחלקה (Fields, Properties) 5.3.1.2

– להלן השדות והמאפיינים המוגדרים במחלקה

Member Type	Member Name	Type	Description
Field	_composer	Composer	Compositor instance that implements a certain composition strategy algorithm
Field	_melodySeed	IList<IBar>	Existing melody that could serve as a seed for new one.
Field	_midiInputFilePath	string	Path to MIDI input file.
Field	_midiInputFileName	string	Name of the input MIDI file.
Field	_melodyTrackIndex	MelodyTrackIndex?	Index of the existing melody in the MIDI file, if such exists.
Property	DefaultMinPitch	NotePitch	Default lowest and highest bounds for pitch range in the composition.
Property	DefaultMaxPitch	NotePitch	
Property	MidiInputFile	IMidiFile	Input MIDI file handle.
Property	MidiOutputFile	IMidiFile	Output MIDI file handle.
Property	ChordProgression	IList<IBar>	Chord progression of the song.
Property	MusicalInstrument	MusicalInstrument	Requested musical instrument.
Property	CompositionStrategy	CompositionStrategy	Enumeration property for holding the requested composition algorithm. In turn, this property determines the value of the _composer private field via a factory.

כפי שניתן להבחין, מחלקה זו כוללת תכונות רבות, ובהתאם יש לה לא מעט תלויות – תלות במנשך המיציג את קובץ ה-MIDI, תלות באלגוריתם הלחנה (Dependencies) (Composition Strategy) ועוד. חלק מתלויות אלו "מושרכות" פנימה באמצעות הבניי, דוגמת קובץ ה-MIDI, ואילו אחרות מושרכות כפרמטרים למתחודות, למשל אלגוריתם הלחנה נשלח כารגוומנט למתחודת הלחנה בפועל, כך שניתן יהיה להשתמש באותו מופע של CompositionContext על שיר מסויים ובכל בקשת הלחנה לספק אלגוריתם הלחנה אחר.

מתחודות המחלקה (Methods) 5.3.1.3

מחלקה מגדרה מספר מתחודות. ניתן לחלק אותן במספר קטגוריות: בדיקת תקינות, קריאת קלט, מתחודות עזר לתהיליך הרכבת קובץ הפלט ומתחודת הלחנה עצמה –

- **מתחודות בדיקות תקינות:** מתחודות אלו מחזירות ערך בוליани המחווה על תוצאת הבדיקה.
– להלן בדיקות התקינות המוגדרות במחלקה

Method	Description
AreBarsCompatible	Checks if the bar sequence description in MIDI file is compatible with the bar sequence description in the chord progression file, in terms of length, content, etc.
IsPitchRangeValid	Checks if the requested pitch range is valid.
IsMelodyTrackIndexValid	Checks if the mentioned index for identifying the existing melody track in the MIDI file is valid.

מетодות קריית קלט: מетодות אלו אחראיות על קריית התכנים מקבצי הקלט של ה-MIDI והאקורדים (ו/או מזרים של בתים במקומות מסוים קובץ) והחזירת "ידיוט" המפנות למופעים של תכנים אלו – במקורה של תוכן MIDI זה פשוט מופיע של הממשק IMidiFile, ובמקרה של תוכן רצף אקורדים הידית היא הפניה לאוסף של תיבות המכילות את האקורדים כפי שמתואר באופן טקסטואלי בקובץ הקלט / זרם הבטים מהקלט. לאחר שפעילות של קריית קלט אינה תלואה במופיע ספציפי של מחלקה קונטקט הלחנה, מетодות אלו הוגדרו כMETHODS סטטיות. להלן שמות המethods ותיאור תמציתית שלהם –

Method	Description
ReadMidiFile	Read MIDI content out of a MIDI file and return a handle to it via an IMidiFile instance.
ReadChordsFromFile	Read a chord progression out of a chords text file and return a bar sequence that represents it.

פורמט קובץ האקורדים: להבדיל מפורטט קובץ MIDI שהינו פורטט המוגדר היבר הודות לפרוטוקול MIDI שהפח לסטנדרט, אין פורטט סטנדרטי אחד לקובץ המכיל תיאור של אקורדים. לפיכך, בפרויקט הוגדר פורטט ייעודי לקלט הטקסטואלי המתאר מחלק אקורדים. פורטט זה מתואר [בסוף תת-סעיף זה](#).

מетодות בדיקות תקינות: מethods אלו מחזירות ערך בוליאני המחווה על תוצאות הבדיקה. להלן בדיקות התקינות המוגדרות במחלקה –

Method	Description
AreBarsCompatible	Checks if the bar sequence description in MIDI file is compatible with the bar sequence description in the chord progression file, in terms of length, content, etc.
IsPitchRangeValid	Checks if the requested pitch range is valid.
IsMelodyTrackIndexValid	Checks if the mentioned index for identifying the existing melody track in the MIDI file is valid.

מетодות עזר: המחלקה מגדרה שתי METHODs האורזיות פעילות שימושית שחוזרת על עצמה – האחת אחראית על יצירת קובץ פלייבק, דהיינו קובץ שמכיל רק ליווי ללא מנגינה ראשית (בדומה לקובץ קריוקי), ואילו השנייה אחראית על שכפול רצף אקורדים –

Method	Description
CreateMidiPlayback	Duplicates a given midi file, removes its melody track, and returns the result (midi playback).
CloneChordProgressionBars	Duplicates a given chord progression and removes any empties out any note sequences from the containing bar sequence.

מетод הלחנה: זהה למmethod שמספקת את שירות הלחנה, כלומר זהה למmethod הראשית שהקליניטים פונים אליה במטרה לקבל מנגינה חדשה –

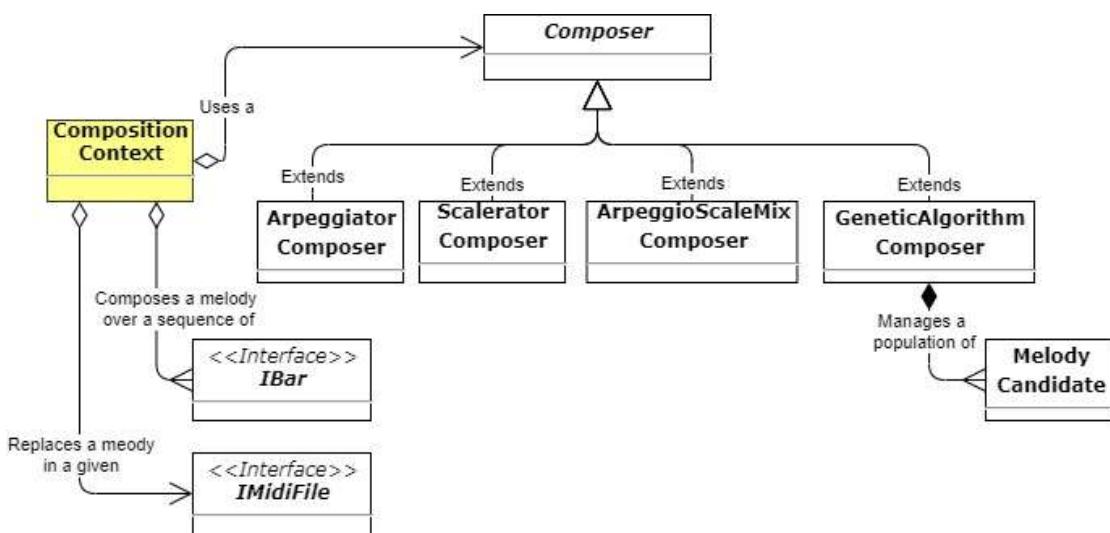
Method	Description
Compose	Composes a solo-melody over this composition's midi playback file and chord progression, using the additional user preferences and constraints parameters.

(Class Diagram) 5.3.1.4

להלן דיאגרמת מחלקה של מחלקה קונטיקסט הלחנה –

CompositionContext
<pre> - _composer: Composer - _melodySeed: IList<IBar> - _melodyTrackIndex: MelodyTrackIndex? - _midiInputFilePath: string - _midiInputFileName: string + ChordProgression: IList<IBar> + CompositionStrategy: CompositionStrategy + DefaultMinPitch: NotePitch + DefaultMaxPitch: NotePitch + MidiInputFile: IMidiFile + MidiOutputFile: IMidiFile + MusicalInstrument: MusicalInstrument + CompositionContext(IList<IBar>, IMidiFile, MelodyTrackIndex?) + CompositionContext(string, string, MelodyTrackIndex?) + AreBarsCompatible(IList<IBar>, IMidiFile, out string): bool + Compose(CompositionStrategy, OverallNoteDurationFeel, MusicalInstrument, PitchRangeSource, NotePitch, NotePitch, bool, params object[]): IMidiFile[] ~ CloneChordProgressionBars(IEnumerable<IBar>): IList<IBar> + CreateMidiPlayback(Stream, MelodyTrackIndex?): IMidiFile + IsMelodyTrackIndexValid(int?, IMidiFile, out string): bool + IsPitchRangeValid(int, int, out string): bool + ReadMidiFile(string): IMidiFile + ReadMidiFile(Stream): IMidiFile + ReadChordsFromFile(string): IList<IBar> + ReadChordsFromFile(StreamReader): IList<IBar> </pre>

להלן דיאגרמה נוספת המתמקדת בקשר שבין מחלקה קונטיקסט ליתר הרכיבים המפורטים בהמשך השיככים לתת-שכבה זו של שירות הלחנת המנגינות –



5.3.1.5 נספח פורמט קלט מהלך האקורדים

- קובץ האקורדים הנתמך בשלב הנוכחי הוא קובץ טקסט. בתחתית הנספח מצורף קובץ שכזה לדוגמה. פורמט תוכן הקובץ הוא כדלקמן –
- הקובץ מורכב משורות המייצגות תיבות, כאשר על כל שורה בקובץ לייצג תיבה בודדת (כל תיבה מיוצגת בשורה נפרדת).
 - על השורות בקובץ להופיע בסדר תואם לסדר התיבות בשיר.
 - מותרות שורות ריקות (שורות רוח) בין שורות התיבות.
 - פורמט כל שורה בקובץ הוא כדלקמן –
 - בראש כל שורה ייכתב המשקל של התיבה בפורמט מונה/מכנה. למשל 3/4 מייצג משקל של שלושה רביעים לתיבה (המונה הוא 3 והמכנה 4).
 - לאחר משקל התיבה תירשם סדרת האקורדים בתיבה מופרדים ביןיהם ברוח אחד או יותר. הפרדה עם רווח אחד או יותר נדרשת גם בין המשקל בראש התיבה לבין האקורד הראשון בתיבה. פורמט כל אקורד הוא כדלקמן –
 - שם התו שמהווה את שורש האקורד, מקו ('-'), שם סוג האקורד, מקו ('-') ומספר הפעימות המהוות את משך השהייה של האקורד בתיבה זו. למשל C-Major7-2 מציין אקורד ששורשו הוא התו C (דו) מסוג מז'ור 7, שנמשך שתי פעימות.
 - שמות תווים שורש האקורדים המותרים לשימושם השמות המקובלים בטרמינולוגיה של המוסיקה המערבית: A עברו לה, B עברו סי, C עברו דו, וכן הלאה עד ל-G עברו סול. בצדี้ לצייןתו עם סימן התק (דיאז או במול), יש להוסיף לתו את הסימנות Sharp או Flat בהתאם. למשל CSharp מציין את התו דו-דיאז #, ואילו BFlat למשל מציין את התו סי-במול Bb.
 - שמות סוגי האקורדים המותרים הם אלו שבעמודה השמאלית בטבלה שלහן (עמודות קוד סוג אקורד) –

תיאור	קוד סוג אקורד
אקורד מוקטן (1-3b-5b-7b)	Diminished
אקורד 7 : מז'ור עם תוספת ספטימה קטנה (1-3-5-7b)	Dominant7
אקורד 7 עם 3 מוחלף ב-4 (1-4-5-7b)	Dominant7Suspended4
אקורד 7 עם דרגה חמישית מוגדלת (1-3-5#-7b)	Dominant7Augmented5
אקורד 7 עם דרגה תשיעית קווקטנת (1-3-5-7b-9b)	Dominant7b9
אקורד 7 עם דרגה תשיעית מוגדל (1-3-5-7b-9#)	Dominant7Sharped9
אקורד חצי-מוקטן (1-3b-5b-7)	HalfDiminished
אקורד מז'ור משולש (1-3-5)	Major
אקורד מז'ור משולש בתוספת הדרגה ה- 6 (1-3-5-6)	Major13
אקורד משולש עם דרגה שלישית מוחלפת ברביעית- 5 (1-3-5-7)	MajorSuspended4
אקורד מז'ור משולש עם דרגה חמישית מוגדלת (#1-3-5-6)	MajorAugmented5
אקורד מז'ור עם תוספת ספטימה גדולה (1-3-5-7)	Major7
אקורד מינור משולש (1-3b-5)	Minor
אקורד מינור משולש בתוספת הדרגה ה- 6 (1-3b-5-6)	Minor6
אקורד מינור בתוספת ספטימה קטנה (1-3b-5-7b)	Minor7
אקורד מינור בתוספת ספטימה גדולה (1-3b-5-7)	MinorMajor7

להלן קובץ אקורדים לדוגמה להמחשת הפורמט שתואר לעיל:



twenty_years_chords.txt

5.3.2 מחלקה מלchein אבסטרקט (Composer Abstract Class)

המחלקה Composer היא מחלקה אבסטרקטית המשמשת כרכיב strategy בדפוס העיצוב Strategy Pattern, בהקשר של אסטרטגייה אלגוריתמית להלחנת מנגינות. המחלקה אחראית על הגדרת מנשך פונקציונאלי משותף לכל המחלקות הקונקרטיות הממששות אלגוריתם הלחנה, וכן לספק תשתיות עזר של שירות הלחנה גנריים בסיסיים שאינם תלויים באלגוריתם הלחנה ספציפי, אלא יהיו שימושיים לכל המחלקות היורשות ויוכלו להוות אבני-בניין לבניית שגרות מורכבות ופרטניות יותר. שירותים כאלה יכולים לכלול למשל הפוך או ערבול סדר של רצף תווים נתון, יצירת רצף אקרייא של מימייה, עדכון גובה צליל בכמה חצאי-טוניים, הארכה/קיצור משך שהייה של צליל ועוד.

המחלקה עצמה היא כאמור אבסטרקטית. זה בא לידי ביטוי בכך שהיא אינה מספקת אלגוריתם הלחנה בעצמה, אלא מגדרה מתודת אבסטרקטית – GenerateMelody – של המחלקות היורשות ממנה למש. כל מחלקה קונקרטית שיורשת Composer מייצגת איזשהו אלגוריתם הלחנה שביצע את הלחנה במסגרת IMPLEMENTATION הלחנה: שלושה מהם אלגוריתמים מאוד בסיסיים מספקת מימושים של ארבעה אלגוריתמי הלחנה: שלושה מהם אלגוריתמים מאוד בסיסיים שפושט מחוללים רצף תווים המבוסס בהתאם על האקורדים, הסולמות המומפיים לאקורדים, וצירוף של השניים, ואילו המימוש הרוביעי, שהוא ה-”מח” ו-”לב” המערכת – הוא מימוש אלגוריתם גנטי, המפורט בהמשך.

5.3.2.1 תכונות המחלקה (Fields, Properties)

להלן השדות והמאפיינים המוגדרים במחלקה –

Member Type	Member Name	Type	Description
Field	PossibleDurationFractions	float[]	Constant duration fractions of the most used durations.
Property	ChordProgression	IList<IBar>	The playback's harmony.
Property	ComposedMelody	IList<IBar>	The outcome of the Compose method.
Property	DefaultDuration	IDuration	Default duration for a single note.
Property	DefaultDurationDenominator	byte	Default duration denominator for a single note.
Property	DefaultDurationFraction	float	Fraction of the default duration for a single note.
Property	LongestAllowedDurationDenominator	byte	Upper bound for a duration's denominator & fraction that is set according to the requested OverallNoteDurationFeel.
Property	LongestAllowedFraction	float	
Property	MaxOctave	byte	Maximum & Minimum octaves of note pitch range for the composition.
Property	MinOctave	byte	
Property	MaxPitch	NotePitch	Highest & lowest bounds of a note pitch for the composition.
Property	MinPitch	NotePitch	
Property	Seed	IList<IBar>	Melody seed to base on the composition of the new melody.
Property	ShortestAllowedDurationDenominator	byte	Lowest bound for a duration's denominator & fraction that is set according to the requested OverallNoteDurationFeel.
Property	ShortestAllowedFraction	float	

5.3.2.2 מетодות המחלקה (Methods)

מетодות עברו שירוטי הלחנה בטיסיטיפ: מетодות אלו משמשות כבניו בינוי בסיסיות לביצוע מניפולציות ועדכונות שונים על רצפי תווים, משכי שהייה, אקורדים ותיבות. בניו בינוי אלו יכולים להיות לעזר בכל פעילות הלחנה ללא תלות באופן המימוש של אלגוריתם הלחנה, ולכן הם מושפעים בחלוקת האב האבסטרקטית כך שכל מחלקה יורשת תוכל לבחור להשתמש בהם ע"פ הצרכים שלה. כל המethodות הללו מוגדרות כ-protected private על מנת שה[method] מוגדרות כ-protected virtual על מנת שהמחלקות בנות המשענויות בכך יוכל לדרוס override את פונקציונאליותם ביחס למוחלקות הוראות וממשימות כבניו בינוי להגדרתן. חלק מmethods השירות שלහן הם עצמן גנריות וממשימות כבניו בינוי להגדרתן מетодות שירותים ספציפיות יותר עם ריאיציות שונות. להלן פירוט כל methods השירות –

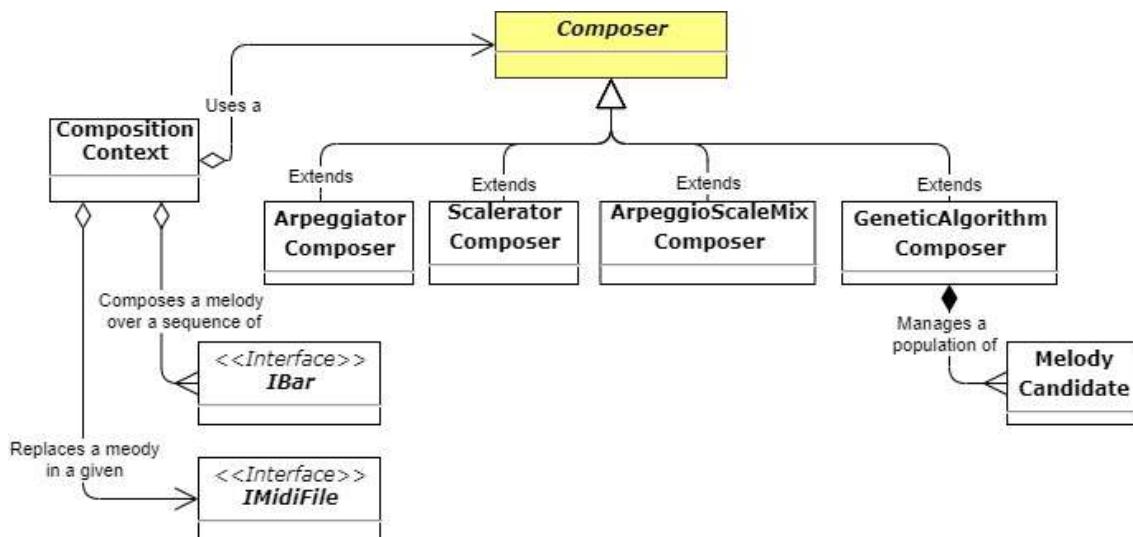
Method	Description
ArpeggiatorInitializer	General arpeggiator initializer that initializes an entire note sequence for a given sequence of bars, based on the bars chord progression arpeggio notes and requested note sequence mode (ascending, descending or zigzag).
ArpeggiatorInitializerAscending	Initializes a note sequence for a given sequence of bars, based on the bars chord progression arpeggio notes either in ascending/ descending order or in alternate order for each chord/bar.
ArpeggiatorInitializerDescending	
ArpeggiatorInitializerChordZigzag	
ArpeggiatorInitializerBarZigzag	
ChangePitchForARandomNote	Selects a random note in a bar and changes its pitch to another note from the requested mapping source (chord/scale).
NoteDurationSplit	Replaces a random note in the given bar with two new shorter notes with durations that sum up together to the original's note duration. Regarding pitch, one of the new notes after split would have the original's note pitch, and the other note would have a pitch which is a minor or major second away from the original pitch.
NoteSequenceInitializer	Generic initialization methods that initializes an entire note sequence for a given sequence of bars, based on custom user preferences such as melody contour direction and a chord-note mapping source.
PermuteNotes	Generates a permutation of the bar's note sequence & replaces the original sequence with the permuted sequence (shuffled/reversed, etc).
ScaleArpeggioeMixInitializer	Initializes a note sequence for a given sequence of bars, based both on the bars chord progression mapped scales & the arpeggio notes.
ScalculatorInitializer	General scalculator initializer that initializes a note sequence for a given sequence of bars, based on the bars chord progression mapped scale notes and requested note sequence mode (ascending, descending or zigzag).
ScalculatorInitializerAscending	Initializes a note sequence for a given sequence of bars, based on the bars chord progression mapped scale notes either in ascending/descending order or in an alternating order with each chord / bar.
ScalculatorInitializerDescending	
ScalculatorInitializerChordZigzag	
ScalculatorInitializerBarZigzag	
SyncopizeANote	Syncope's a bar's first note by connecting it to its preceding note (last note from preceding bar).
ToggleAHoldNote	Replaces a random hold note with a concrete note pitch.

מетодות אתחול והלחנה: המתוודות שלහן נמצאות בשימוש ע"י כל המחלקות היורשות: הראשונה ([InitializeCompositionParams](#)) אחראית על אתחול שבוצע ממש לפני פעילות הלחנה. מותוודת אתחול זו מוגדרת כ-*virtual* לצורך האפשרות למחלקות יורשות לدرس אותה ([override](#)) ולמשתמש התאמות ע"פ צרכים שלהם. המתוודה השנייה ([GenerateMelody](#)) היא המתוודה האבסטרקטית שמיועדת להכיל את הלוגיקה העסוקה לМИוש אלגוריתם הלחנה, והמוודת השילישית היא שזו שהקלינייטים מחוץ למחלקה מכיריהם. היא עוסקת את מותוודת האתחול ומותוודת הלחנה ה-"אמיתיית" ([GenerateMelody](#)) שאחרראית על ביצוע הלחנה בפועל. להלן פירוט נוסף –

Method	Description
InitializeCompositionParams	Initialize general parameters such as duration default values and bounds, pitch range, melody seed, chord progression etc. This method is defined as virtual so that it could be overridden by sub-classes to implement custom-specific initializations. Additional parameters could be sent via the 'params object[] additionalParams' argument. This method is called right before the composition takes place.
<i>GenerateMelody</i>	Abstract method that is responsible to carry out the actual composition process and return a melody to the caller. Each subclass has to implement this method by supplying the business logic that processes all the composer's data and This method is called right after the InitializeCompositionParams method is done.
Compose	The method that is exposed to the class clients as an end point for requesting a composition. Internally this method wraps the flow structure: It is responsible for calling the initialization method for the initialization phase, and then executing the actual composition method for generating the melody.

5.3.2.3 דיאגרמות מחלקה

להלן דיאגרמות מחלקה המציגות את הקשר בין מחלקות המלחין האבסטרקטית ליתר הישויות בתת-שכבה זו, וכן את כל המאפיינים והמוודות המוגדרות במחלקה –



– דיאגרמת מחלקה המכילה את כל המאפיינים והmethod'ות המוגדרות במחלקה הקונטקט –

<i>Composer</i>
<pre> -#PossibleDurationFractions: float[] ~ ComposedMelody: IList<IBar> ~ ChordProgression : IList<IBar> ~ ComposedMelody: IList<IBar> ~ DefaultDuration: IDuration ~ DefaultDurationDenominator: byte ~ DefaultDurationFraction: float ~ DefaultNumOfNotesInBar: byte ~ LongestAllowedDurationDenominator: byte ~ LongestAllowedFraction: float ~ ShortestAllowedDurationDenominator: byte ~ ShortestAllowedFraction: float ~ MinOctave: byte ~ MaxOctave: byte ~ MinPitch: NotePitch ~ MaxPitch: NotePitch -# GenerateMelody(): IEnumerable<IList<IBar>> ~ Compose(IList<IBar>, IList<IBar>, OverallNoteDurationFeel , NotePitch, NotePitch, object[]): IEnumerable<IList<IBar>> -# InitializeCompositionParams(IList<IBar>, IList<IBar>, OverallNoteDurationFeel, NotePitch, NotePitch, object[]): void -# NoteSequenceInitializer(IEnumerable<IBar>, NoteSequenceMode, ChordNoteMappingSource, bool): void - GenerateDurations(float, float?): IDuration[] -# ArpeggiatorInitializer(IEnumerable<IBar>, NoteSequenceMode): void -# ArpeggiatorInitializerAscending(IEnumerable<IBar>): void -# ArpeggiatorInitializerDescending(IEnumerable<IBar>): void -# ArpeggiatorInitializerChordZigzag(IEnumerable<IBar>): void -# ArpeggiatorInitializerBarZigzag(IEnumerable<IBar>): void -# ScaleratorInitializer(IEnumerable<IBar>, NoteSequenceMode): void -# ScaleratorInitializerAscending(IEnumerable<IBar>): void -# ScaleratorInitializerDescending(IEnumerable<IBar>): void -# ScaleratorInitializerChordZigzag(IEnumerable<IBar>): void -# ScaleratorInitializerBarZigzag(IEnumerable<IBar>, NoteSequenceMode): void -# ScaleArpeggioeMixInitializer(IEnumerable<IBar>, NoteSequenceMode): void -# ChangePitchForARandomNote(IBar, ChordNoteMappingSource, byte): bool -# NoteDurationSplit(IBar, DurationSplitRatio): bool -# PermuteNotes(IBar, IEnumerable<IChord>, Permutation): void -# ToggleAHoldNote(IEnumerable<IBar>, int?): bool -# SyncopizeANote(IList<IBar>, int?): bool -# NoteDurationSplit(IBar, DurationSplitRatio): bool -# PermuteNotes(IBar, IEnumerable<IChord>, Permutation): void -# ToggleAHoldNote(IEnumerable<IBar>, int?): bool </pre>

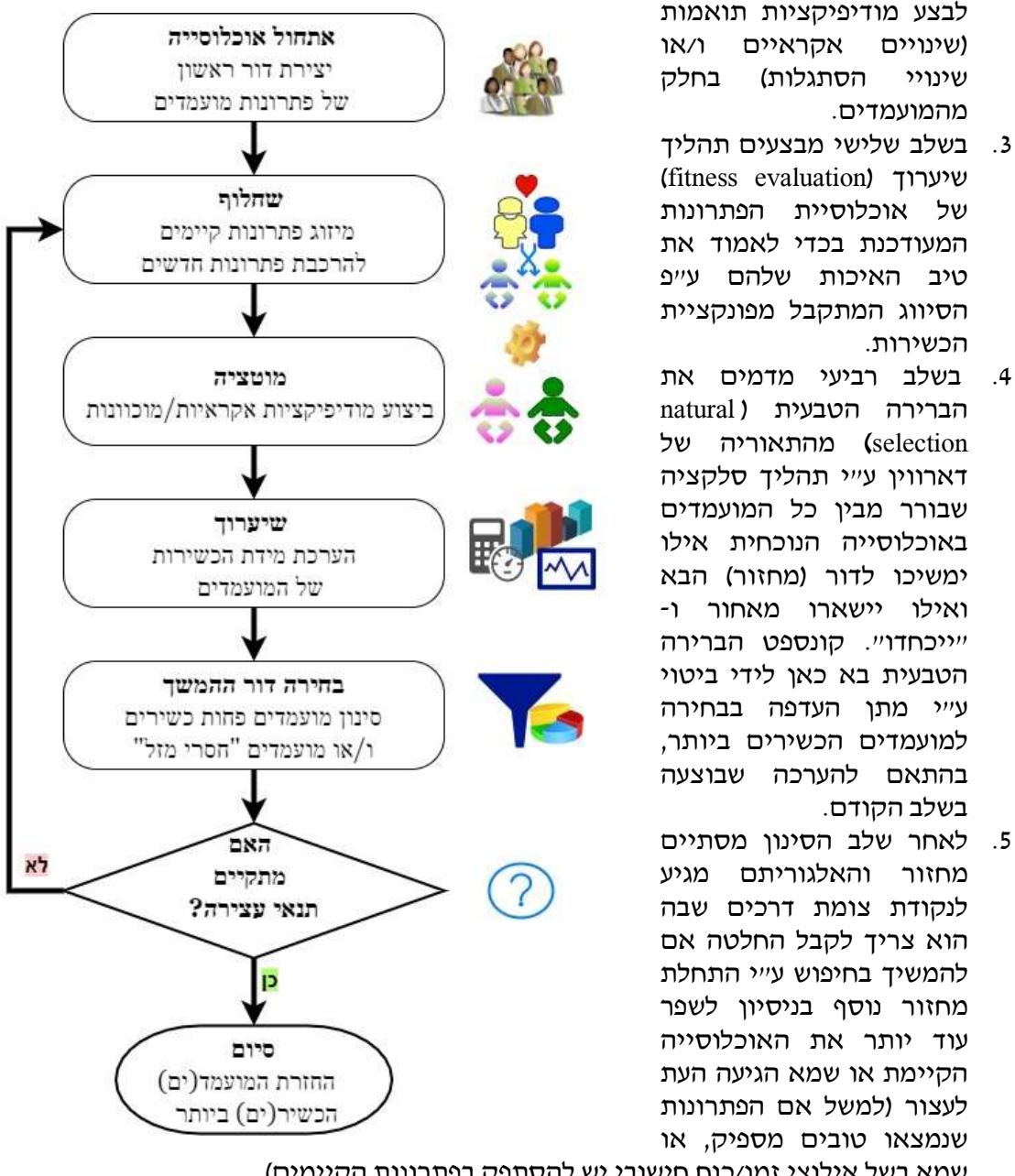
5.3.3 הלחנה עם אלגוריתם גנטי

5.3.3.1 רקע לאלגוריתמים/genetic

משפחת האלגוריתמים הגנטיים היא אוסף שיטות אופטימיזציה מטא-היוריסטיות המבוססות על עקרונות שאומצו בהשראת תהליכי אבולוציוניים מהביולוגיה. התבנית הכללית לאלגוריתמים אלו היא דמיון מלאכותי של אבולוציה על אוכלוסיות פתרונות מועמדים ע"י תהליך איטרטיבי: בכל איטרציה מבוצע עדכון ומיזוג של הפתרונות המועמדים, שיעורץ טיב איכות פתרונות אלו ותהליך סלקציה המדמה את עיקרו "החזק שורד" לבחירת תת-קבוצת הפתרונות המועמדים שתמשיך לאיטרציה הבאה.

תיאור שלבי אלגוריתם גנטי טיפוסי: האלגוריתם מתנייע את תהליך האבולוציה המלאכותי עם אתחול דור ראשון של פתרונות מועמדים, שנitinן לחולל זה באופן אקראי, והן באופן יזום על-סמן פרמטרים נבחרים. על בסיס אוכלוסייה זו של הדור הראשון מתחילה מחזורי (מעגלי) המדמה תהליכי אבולוציוניים מהביולוגיה (ראה איור משמאל):

1. בשלב ראשון מגדמים תהליכי של זיווג/רביה ע"י שימוש באופרטור השחלוף שמנוג חלקיים שני פתרונות או יותר (ההורים) ובונה מהם פתרונות חדשים (צאאים).
2. בשלב שני מגדמים "רעש" שקיים בטבע כגון מוטציות גנטיות והשפעות סביבתיות הקופות שינוי הסתגלותי (אaptive) על המועמד. שלב זה ממומש ע"י אופרטור מוטציה שאחראי



להלן פסידו-קוד של האלגוריתם שתואר לעיל –

```

1. מתחם זיכר כלכלי
2. גזע -
  ↗יינט 2.1
  ↗אינט 2.2
  ↗עיצלאם 2.3
  ↗מחילת זיכר הנק
3. אפקט האם מתקיים תỰק'י הצעירה -
  ↗אם כן, המצלב כפוף את האנציאן(ים) הכהיל(ים) פותח וסימן;
  ↗אחרת, צוואר ספה 2.3.2
  ↗אחרת, צוואר ספה 2.3.1

```

בקשר של המערכת הנידונה, בעיתת האופטימיזציה הנתונה היא חיבור מנגינה, אוכלוסיית הפתורונות המועמדים היא קבוצת מניגנות שmotocn על האלגוריתם למצוא את זו "הטובה" ביותר, דהיינו זו שנשמעות הכוי "טוב". מה זאת מנגינה טובה? זו שאללה די סובייקטיבית, אולי ניתן להגיד מודדים כליליים שונים ולתת למשתמש (המאזין) את יכולת לקבועיחס סדר בין מודדים אלו, כך שהאלגוריתם יעדיף מניגנות שמצטיינות במודדים שהמשתמש מייחס להם חשיבות גבוהה יותר. זאת, יחד עם מתן האפשרות למשתמש לצין העדפות נוספות כגון מידת דחיסות הצלילים ומונענד נתנות איזשהו קירוב לממה שהמשתמש יכול להחשיב כמנגינה טובה בעניינו (או ליתר דיוק, באזונו).

פירוט נוסף על כל אחד משלבי האלגוריתם שלעיל, על אלגוריתמים גנטיים בכלל ודיוון בישום שלהם לחיבור מניגנות, ניתן למצוא [במסמך המצורף לעבודת סמינר בנושא](#)



Genetic Algorithms For
Melody Composition

5.3.3.1 תיאור ו McCabe הפיתוח של האלגוריתם הגנטי

חלוקת מחלקה האלגוריתם הגנטי למודולים

כל אחד מהשלבים של האלגוריתם הגנטי שתוארו לעיל (אתחול, שחלוּף, מوطיצה, וכו') מכילים לא מעט לוגיקה. בכך לשמר על סדר בחלוקת האלגוריתם הגנטי (GeneticAlgorithmClass), אשר כל קובץ אחראי המחלוקת פועל סביר מספר קבצים (כל קובץ מוגדר כ-partial class), כל מודול מגדר מותודות ניהול ראשית שנקראות מותוך לטפל באחד המודולים של האלגוריתם. כל מודול מגדר מותודות ניהול פרטיים הטעניים. להלן תקציר האלגוריתם הגנטי המרכזי, ומתחוזות עזר פנימיות שמטפלות בפרטים הטעניים. להלן תקציר תוכן הקבצים (מודולים) השונים שמגדירים ייחודי את המחלוקת. GeneticAlgorithmClass, עם תקציר התוכן שלהם שם מותודות ניהול שלהם (הмотודה שנקרית מותוך האלגוריתם המרכזי) –

תוכן	מזהות ניהול מרכזי	שם קובץ (מודול)
הגדרת תזרים האלגוריתם הגנטי.	GenerateMelody	GeneticAlgorithmComposer.cs
אתחול דור ראשון של מניגנות מועמדות.	PopulateFirstGeneration	Initialization.cs
שלב השחלוף של האלגוריתם.	Crossover	Crossovers.cs
שלב המوطיצה של האלגוריתם.	Mutate	Mutators.cs
שלב הערכת טוב איקוט המנגינות מועמדות.	EvaluateFitness	FitnessEvaluators.cs
שלב בחירת המנגינות שימושיו לדור הבא.	SelectNextGeneration	Selectors.cs

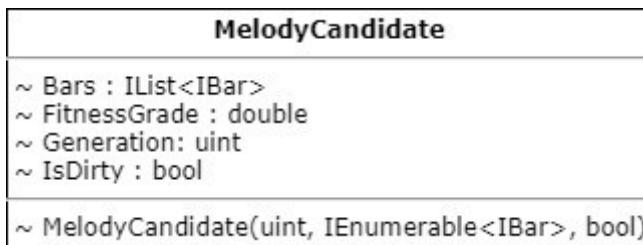
בנוסף לחלוקת האלגוריתם הגנטי (GeneticAlgorithmClass), הוגדרה מחלוקת נפרדת לייצוג מנגינה אינדיוידואלית המשתתפת כמנגינה מועמדת באלגוריתם הגנטי : MelodyCandidate .

העמודים הבאים מכילים תיעוד מפורט על כל אחת מחלוקתות אלו והמודולים השונים.

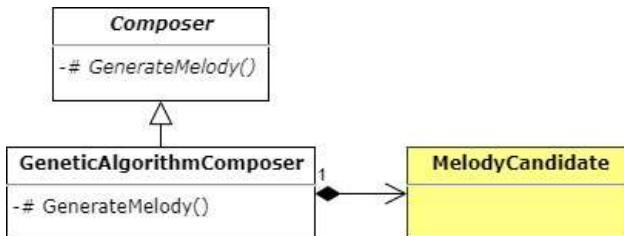
MelodyCandidate Class 5.3.3.2

מחלקה זו מייצגת פתרון מועמד של מנגינה באלגוריתם הגנטי. מלבד המנגינה עצמה (רצף תיבות המכיל את אוסף הצלילים המרכיבים את המנגינה), כל פתרון מועמד (דיהינו כל מופע של MelodyCandidate) מכיל גם מאפיינים הרלוונטיים לאלגוריתם הגנטי כגון הדור שאליו שייך המועמד (דיהינו מספר האיטרציה באלגוריתם הגנטי שבה המועמד נוצר) וציון מד האיכות שלו. הפתרונות המועמדים השונים נוצרים ע"י מופע כלשהו של אלגוריתם גנטי, ומהזור החיים שלהם מוכל ותלי במחזור החיים של האלגוריתם הגנטי שיצר אותם.

להלן דיאגרמת מחלקה המציגה את המאפיינים והשיטות של המחלקה –



להלן דיאגרמת מחלקה המתמקדת בקשר של מחלקה זו עם מחלקות אחרות –



מאפיינים : להלן תיאור מאפייני המחלקה (Properties) מדיאגרמת המחלקה שלעיל –

Member Name	Type	Description
Bars	IList<IBar>	List of bars which contain the melody of this candidate.
FitnessGrade	double	The current score of this candidate, which signifies how good it is.
Generation	uint	The generation that this candidate belongs in, i.e., what generation was he created at.
IsDirty	bool	Flag which indicates whether this candidate was modified during the last iteration of the genetic algorithm operating on it. This flag could be used to utilize performance by skipping fitness evaluation for unmodified (i.e., clean, un dirty) candidates.

בנייה : המחלקה מדירה בנאי יחיד המתחול מועמד בהינתן לו מספר הדור (איטרציה נוכחית של האלגוריתם) שאליו הוא משתמש, המבנה הרמוני שלו יש לייצר מנגינה (רצף אקורדים המוכל ברצף התיבות הנתון בקלט) ופרמטר הקבוע האם לאתחול את המועמד החדש עם המנגינה הקיימת ברצף התיבות או ליצור את המועמד עם מנגינה ריקה –

```
internal MelodyCandidate(uint generation, IEnumerable<IBar>
compositionStructure, bool includeExistingMelody = false) {...}
```

הדור של המועמד נשאר קבוע לכל אורך מחזור החיים שלו. לעומת זאת, הציון (FitnessGrade) שלו משתנה בין האיטרציות של האלגוריתם הגנטי המכיל אותו בהתאם לשינויים החלים על המנגינה שהמועמד מייצג, ופונקציית השערוך שאומדנת מנגינה זו.

GeneticAlgorithmComposer Class 5.3.3.3

המחלקה GeneticAlgorithmComposer יורשת מהמחלקה האבstarטקטית Composer ממסמת את אסטרטגיית הלחנה שלה בגוף המתוודה האבstarטקטית GenerateMelody תוך שימוש **מבנה פסידו-קוד הכללי של אלגוריתמים גנטיים** שהוצגה לעיל. להלן סגמנט מהקוד של פימוש פסידו-קוד זה המוגדר במודול הראשי של האלגוריתם הגנטי, דהיינו בקובץ – GeneticAlgorithmComposer.cs

```
private protected override IEnumerable<IList<IBar>> GenerateMelody()
{
    PopulateFirstGeneration(); // generate first generation

    int i = 0;
    bool terminateCondition = false

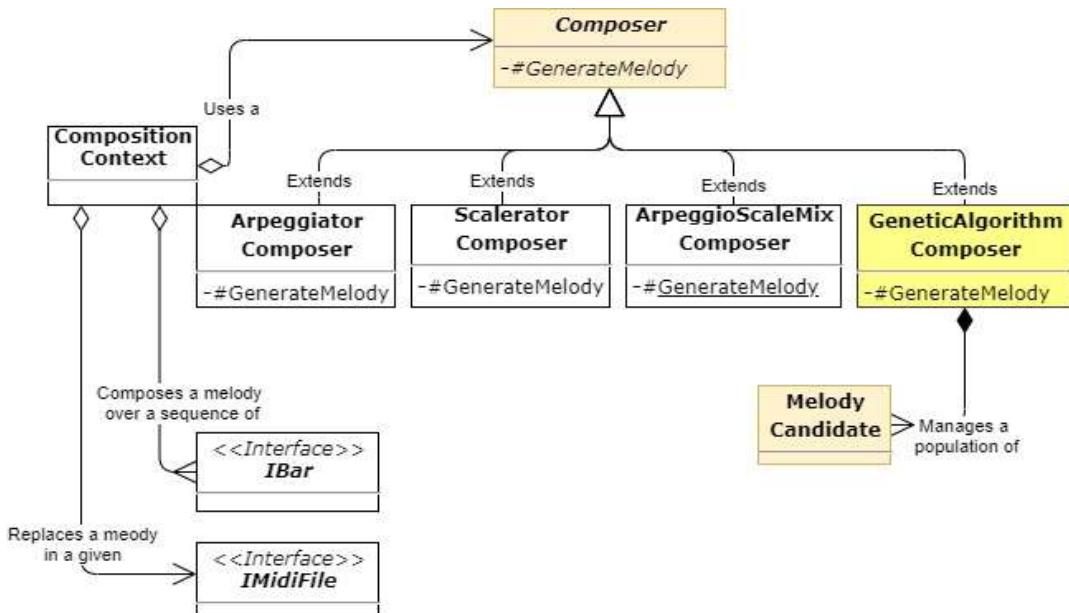
    while (!terminateCondition)
    {
        _currentGeneration++;
        Crossover(); // mix & combine different individuals
        Mutate(); // modify parts of individuals
        EvaluateFitness(); // rate each individual
        SelectNextGeneration(); // natural selection

        if (++i == MaxNumberOfIterations || ... )
            terminateCondition = true;
    }
    return ...
}
```

הMATODEOT הפנימיות הן מתודות ניהול המוגדרות במודולים השונים שיחדיו מרכיבים את המחלקה כפי שתואר שליל (כל אחד מהם מוגדר כ-partial class). פירוט כל אחד ממודולים אלו נמצא בסעיפים הבאים.

מעבר למתוודה Composer, במודול זה מוגדרת רק עוד מתוודה אחת יחידה: InitializeCompositionParams. המתקבלת בירושה ממחלקת המלחין האבstarטקטית Composer ואחרה-יאת על אתחול ראשון של פרמטרים לאלגוריתם הלחנה. האלגוריתם הגנטי מנצל מותוודה זו לאייפוס מספר הדור הגלובאלי לkrarat מחזור ריצה חדש, אתחול אוכלוסיית המנגינות המועמדות ואתחול משקלים יחסיים לפונקציות השערוך (פירוט נוסף על משקלים אלו בפרק המתאר את [מודול השערוך](#)).

להלן דיאגרמת מחלקה המדגישה את הקשר של מחלקות האלגוריתם הגנטי ביחס ליתר היחסויות בתת-שכבה זו –



להלן דיאגרמת מחלקה (class diagram) עם כל המאפיינים והmethod'ות במחלקה זו –

GeneticAlgorithmComposer
<pre> -# _barMutations : Dictionary<Action<MelodyCandidate, int?>, double> -# _candidates : List<MelodyCandidate> -# _currentGeneration : uint -# _initializers : Action<IEnumerable<IBar>>[] -# CuttingEvaluationGrade : double -# MinNumberOfIterations : int -# MaxNumberOfIterations : int -# MaxPopulationSize : int -# MinMutationProbability : double -# MutationProbabilityStep : double ~# EvaluatorsWeights : MelodyEvaluatorsWeights ~-# PossibleDurationFractions: float[] ~~ ComposedMelody: IList<IBar> ~~ ChordProgression : IList<IBar> ~~ ComposedMelody: IList<IBar> ~~ DefaultDuration: IDuration ~~ DefaultDurationDenominator: byte ~~ DefaultDurationFraction: float ~~ DefaultNumOfNotesInBar: byte ~~ LongestAllowedDurationDenominator: byte ~~ LongestAllowedFraction: float ~~ ShortestAllowedDurationDenominator: byte ~~ ShortestAllowedFraction: float ~~ MinOctave: byte ~~ MaxOctave: byte ~~ MinPitch: NotePitch ~~ MaxPitch: NotePitch + GeneticAlgorithmComposer() -# ChordPitchMutation(MelodyCandidate, int?) : void # Crossover() : void -# DurationAnticipationSplitMutation(MelodyCandidate, int?) : void -# DurationDelaySplitMutation(MelodyCandidate, int?) : void -# DurationEqualSplitMutation(MelodyCandidate, int?) : void -# DurationSplitMutation(MelodyCandidate, int?) : void -# DurationUnifyMutation(MelodyCandidate, int?) : void ~-# EvaluateFitness() : void -# EvaluateAccentedBeats(MelodyCandidate) : double -# EvaluateContourDirection(MelodyCandidate) : double -# EvaluateContourStability(MelodyCandidate) : double -# EvaluateDensityBalance(MelodyCandidate) : double -# EvaluateExtremeIntervals(MelodyCandidate, PitchInterval) : double -# EvaluateSmoothMovement(MelodyCandidate, PitchInterval) : double -# EvaluateSyncopation(MelodyCandidate) : double -# EvaluatePitchVariety(MelodyCandidate) : double -# EvaluatePitchRange(MelodyCandidate) : double -# Mutate() : void # NPointCrossover(IList<MelodyCandidate>, int, bool) : ICollection<MelodyCandidate> -# PlusSelection() : void -# RouletteWheelSelection() : void # RegisterInitializers() : void -# RegisterMutators() : void -# ReverseAllNotesMutation(MelodyCandidate) : void -# ReverseBarNotesMutation(MelodyCandidate, int?) : void -# ReverseChordNotesMutation(MelodyCandidate, int?) : void -# ScalePitchMutation(MelodyCandidate, int?) : void # SelectOptimizedCrossoverPoints(MelodyCandidate, MelodyCandidate, int) : int[] # SelectRandomCrossoverPoints(int, int) : int[] -# SwapTwoNotesMutation(MelodyCandidate, int?) : void -# SyncopatedNoteMutation(MelodyCandidate, int?) : void -# ToggleFromHoldNoteMutation(MelodyCandidate, int?) : void -# ToggleToHoldNoteMutation(MelodyCandidate, int?) : void # PopulateFirstGeneration : void ~-# GenerateMelody(): IEnumerable<IList<IBar>> ~~ Compose(IList<IBar>, IList<IBar>, OverallNoteDurationFeel, NotePitch, NotePitch, object[]): IEnumerable<IList<IBar>> ~-# InitializeCompositionParams(IList<IBar>, IList<IBar>, OverallNoteDurationFeel, NotePitch, NotePitch, object[]): void ~-# NoteSequenceInitializer(IEnumerable<IBar>, NoteSequenceMode, ChordNoteMappingSource, bool) : void ~-# GenerateDurations(float, float?) : IDuration[] ~-# ArpeggiatorInitializer(IEnumerable<IBar>, NoteSequenceMode) : void ~-# ArpeggiatorInitializerAscending(IEnumerable<IBar>) : void ~-# ArpeggiatorInitializerDescending(IEnumerable<IBar>) : void ~-# ArpeggiatorInitializerChordZigzag(IEnumerable<IBar>) : void ~-# ArpeggiatorInitializerBarZigzag(IEnumerable<IBar>) : void ~-# ScaleratorInitializer(IEnumerable<IBar>, NoteSequenceMode) : void ~-# ScaleratorInitializerAscending(IEnumerable<IBar>) : void ~-# ScaleratorInitializerDescending(IEnumerable<IBar>) : void ~-# ScaleratorInitializerChordZigzag(IEnumerable<IBar>) : void ~-# ScaleratorInitializerBarZigzag(IEnumerable<IBar>, NoteSequenceMode) : void ~-# ScaleArpeggioeMixInitializer(IEnumerable<IBar>, NoteSequenceMode) : void ~-# ChangePitchForARandomNote(IBar, ChordNoteMappingSource, byte) : bool ~-# NoteDurationSplit(IBar, DurationSplitRatio) : bool ~-# PermuteNotes(IBar, IEnumerable<IChord>, Permutation) : void ~-# ToggleAHoldNote(IEnumerable<IBar>, int?) : bool ~-# SyncopateANote(IList<IBar>, int?) : bool ~-# NoteDurationSplit(IBar, DurationSplitRatio) : bool ~-# PermuteNotes(IBar, IEnumerable<IChord>, Permutation) : void ~-# ToggleAHoldNote(IEnumerable<IBar>, int?) : bool </pre>

5.3.3.4 תכונות המחלקה

להלן תקציר על השדות והמאפיינים המוגדרים במחלקה (אלו שאינם מוגדרים במחלקת האב) –

Member Type	Member Name	Type	Description
Field	_barMutations	Dictionary<Action<MelodyCandidate, int?>, double>	Delegates of melody mutation methods and their probability to execute.
Field	_candidates;	List<MelodyCandidate>	The candidate generated melodies that are competing to be selected.
Field	_currentGeneration;	uint	Current iteration of the genetic algorithm.
Field	_initializers	Action<IEnumerable<IBar>>[]	Delegates of melody initialization methods.
Field	CuttingEvaluationGrade	double	A grade value that is high enough for ending algorithm's execution.
Field	MinNumberOfIterations	int	Low & upper bounds of iterations for the algorithm to execute.
Field	MaxNumberOfIterations	int	
Field	MaxPopulationSize	int	Upper bound of the melody candidate population.
Field	MinMutationProbability	double	Low bound of probability for a mutation method to execute.
Field	MutationProbabilityStep	double	Step size between iterations for reducing probability of executing a mutation method.
Property	EvaluatorsWeights	MelodyEvaluatorsWeights	Evaluation methods set of proportional weights for their significance.

5.3.3.5 מודול אתחול (Initialization.cs)

רקע: מודול זה מרכז את פעילות אתחול אוכלוסיית המנגינות הראשונית, דהיינו את הדור הראשון של המנגינות המהוות את הבסיס שעליו יפעל התהליך המחזורי של האלגוריתם. המודול מוגדר בקובץ Initialization.cs. בהתאם לפרמטר אופציונלי, האתחול יכול לבסס את האתחול על סמך מנגינת בסיס כלשהו (seed initialization), למשל – המנגינה המקורית, או אתחול מ-”scratch” שאינו מבוסס כלל על מנגינה קודמת אלא אך ורק על המהלך הרגמוני (אקורדים).

מетодות אתחול: האתחול מבוסס על אבני בניין – מетодות אתחול המוגדרות במחלקה האב Composer. מетодות אתחול אלו מחוללות רצפי תווים על סמך האקורדים של המנגינה. השוני ביןיהן הוא במקור המיפוי של התווים (האם מトוק צלילי האקורד או סולם שmmoפה אל מול האקורד), ובסדר התווים שביצף (האם הרץ' בכל תיבה צריך להיות בעלייה, ירידת או משתנה לסירוגין).

מетодת רישום: המתודה RegisterInitializers אחראית על רישום מетодות האתחול השונות (אבני הבניין). היא נקנית מותוק בנאי המחלקה ורושמת את מетодות האתחול ברשימה ייועדת המוצגת ע"י המאפיין _initializers.

מетодת ניהול האתחול: המתודה PopulateFirstGeneration אחראית על ניהול כל האופורציה סביר האתחול: קריית מетодות האתחול מותוק הרשימה הייועדי, הפעלתן זו אחר זו בסדר להפיק מנגינות שונות, ביצוע מניפולציות ועיבוד נוספים בכדי לייצר וריאציות נוספות וב民意ת הצורך לוחצת בחשבון גם מנגינת בסיס גרעין (seed) ליחס. מתודה זו היא המתודה הנקראת מותוק האלגוריתם הגנטי (דהיינו מותוק המתודה GenerateMelody) לטובת ביצוע האתחול. בסיום פעולה המאפיין _candidates המכיל את אוכלוסיית המנגינות המועמדות השונות מואוכלס וממלא.

מוסכמות חתימה: מетодות האתחול הן בעלות המוסכמה הבאה של חתימה איחידה –
IEnumerable<IBar> bars

כל מетодת אתחול מקבלת רצף תיבות ופועלת במקום (inplace) על רצף תיבות זה ללא החזרת פלט. המוסכמה על איחדות בחתימה מספקת גנריות ו-**דיינמיות**: מетодת ניהול האתחול אינה צריכה להכיר את המethodות האתחול הקונקרטיות השונות כל עוד הן רשומות ברשימה מетодות האתחול _initializers ע"י מетодת הרישום RegisterInitializers. במידה ויהיה צורך לעשות שימוש במетодת אתחול בעלת חתימה אחרת, רצוי יהיה להקים רשימה נוספת delegates עם חתימות תואמות, ולעבוד מותוק אותה רשימה באופן גנרי בשלב הרישום בבניי ואחזרה במетодת הניהול ולא לפנות למетодת אתחול כזו או אחרת באופן מפורש.

תקציר המethodות במודול: להלן המethodות המוגדרות במודול זה (METHODES אתחול של אבני הבניין – מוגדרות כאמור מוחז למודול במחלקה האב Composer)

Method	Description
PopulateFirstGeneration	Initialize first generation of solution candidates.
RegisterInitializers	Register initialization methods that will be called upon initialization of the first generation of candidate melodies.

תיעוד מפורט ומלא של המethodות השונות נמצא בגוף קבצי קוד המקור.

5.3.3.6 מודול שחלוֹף (crossover.cs)

רקע : מודול זה מרכז את פעילות השחלוף בקובץ **“يיעודי : crossover.cs”**. המודול אחראי על יימוש אופרטור השחלוף לאלגוריתם הגנטי. אופרטור זה מצילב וממזג זוגות פתרונות (מנגינות) ומפיק מהם פתרונות חדשים – צאצאים, המורכבים מערבוב רכיבים של הפתרונות שייצרו אותם – **“ההורים”**.

מетодות השחלוף : המתוודה Crossover היא המתוודה שנקראת ישירות מתוך האלגוריתם הגנטי עצמו (דהיינו ישירות מתוך המתוודה [GenerateMelody](#)). מתוודה זו קוראת בתורה למתוודה אחרת NPointCrossover המממשת שחלוֹף בשיטה הפוולרית point-crossover *n*-point crossover המשלבת חלקיק פתרונות מה모עמדים ההורים לשירוגין ע”פ *n* נקודות הצלבה, כאשר *n* ניתן כפרמטר. בהקשר של מיזוג בין מנגינות, הנקודות מייצגות תיבות. אופן בחירת גן הנקודות הצלבה ממומש במתוודות עוזר, האחת בוורתה את הנקודות אקראית, והשנייה מבצעת בחירה אופטימאלית, במובן שנקודות הפייצול נבחרות באופן שמבטיח מעבר “חلك” במידת הניתן בין תווים שכנים בנקודות הפייצול, וע”י כך מניעת מצב שלאחר הפיצול תיווצר מנגינה עם קפיצות חריגות בין צלילים שכנים כתוצאה מחיבור חלקיקי מנגינות “לא קשרות” בגבהים שונים. להלן – תקציר המתוודות במודול זה, פירוט ותיעוד מלא נמצא בגוף קבצי קוד המקור

Method	Description
Crossover	Slices and mixes solution candidates and generates new solutions which are the outcome offspring of the candidates that participated in the crossover process of slice and mix.
NPointCrossover	Implements a crossover between two or more candidate participants in <i>N</i> distinct points.
SelectOptimizedCrossoverPoints	Utility method for selecting crossover points in a way that minimizes the interval outcome the transition point after the crossover.
SelectRandomCrossoverPoints	Utility method for selecting <i>n</i> distinct crossover points randomly.

הערה : בימוש הקלאסי של שיטת הצלבה זו, המיזוג מבוצע בין זוג פתרונות (שני פתרונות בלבד), אולם השימוש הנוכחי מכיל את השימוש הקלאסי ומאפשר מיזוג אוסף כלשהו של פתרונות.

5.3.3.7 מודול מוטציה (mutators.cs)

רקע: סעיף זה מתאר את המודול האחראי על ביצוע מוטציות (שינויים) למנגינות. מוחלז זה מרוכז בקובץ mutators.cs. המוטציות מדומות שינויים הסתגלותיים אבולוציוניים ו- "רעים" הקיימים בטבע, ומעבירים את המנגינות השונות תחת סדרות של עדכונים (לטוב או לרע). המוטציות בעצם מהוות צעדים למרחב החיפוש. כל שינוי לוקח את החיפוש לכיוון מסוים, והשאיפה היא שלאורך זמן, תוך שיטוף פעולה עם תהליכי השערוך והסלקציה, שדוגמים לסן את המנגינות שעברו שינויים לקחו אותם לכיוונים פחות מוצלחים ולהשאיר את אלו שהשינויים עשו להם טוב.

תכונות ורמות שינוי: שינוי להיות שינוי ברמת התו הבודד (כגון שינוי גובה צליל) או ברמת חצי תיבה/TİBBה שלמה (למשל ערבות סדר התווים בחצי התיבה הראשונה). חלק מהשינויים הם ניטרליים, במובן שהם פועלים על המנגינות השונות באופן אקראי ללא מודעות אם השינוי ישבור או ירע את דירוג טיב האיכות, וחלק מהשינויים הם מוכוונים יותר ומוטים בכיוון מסוים בצד לשפר איזושהי מגעה ספציפית שאורתה.

סקלבייליות: גישה פופולרית באלגוריתמים גנטיים היא להתחילה את החיפוש לאחר הਪתרונות האופטימליים בחיפוש לרוחב, ובהמשך לעבור לחיפוש לעומק, דהיינו חיפוש שמכסה שטח רחב היקף אך סורק אותו רק באופן שטхи, ובהמשך לאחר קצר לימוד של השיטה והסקת השערות לגבי אזורים שנראים קצט יותר אטרקטיביים, לצמצם את החיפוש לאוטם אזורים אך לבצע בהם חיפוש יסודי. בכך לאפשר לאלגוריתם להתאים את היקף החיפוש באופן דינامي בזמן ריצה, לכל מוטציה מוקצה ערך ממשי בין 0 ל-1 שמייצג את ההסתברות לביצוע השינוי על מנתנה בתונה באיטרציה מסוימת של האלגוריתם. בנוסף מוגדרים פרמטרים שימושיים לשינויים המנגינה בהתאם לנסיבות התקדמות האלגוריתם. בכך לתמוך בהגשה הפופולרית שלילית, ההסתברויות מופחתת בכל איטרציה מה שمبיא לארוך זמן לפחות ופחות שינויים והתקנסות/התיכזבות לקראת עצירה. המוטציות וההסתברויות המוקצחות להן באתחול מוגדרות בטבלת Dictionary במאפיין _barMutations, כאשר המפתח הוא Action (למלה אחריות על ביצוע המוטציה, והערך הוא ההסתברות של המוטציה להתבצע. הטבלה נבנית במתודת RegisterMutators שמודעת מותך הבנייא. מדרגת השינוי הנוכחית היא קבועה לכל המוטציות והיא מוגדרת במשתנה הקבוע MutationProbabilityStep.

אופן ביצוע המוטציות: כל מוטציה מוגדרת במתודה `יעודית` האחראית על ביצוע שינוי ספציפי. מתודת ניהול המוטציות – `Mutate`, שנראית מותך האלגוריתם הגנטי, אחראית על תפעול וביצוע המוטציות השונות. ראשית היא בוחרת את האוכלוסייה שעליה היא תבצע מוטציות: בכך לשמר איזושהי יציבות, לא כל המנגינות עוברות שינוי בכל איטרציה, אלא רק אוכלוסייה חלקית, חלקה מרכיבת מנגינות חדשות שנוצרו בשלב השחלוף באיטרציה הנוכחית, וחלקה מנגינות "וותיקות" שנוצרו באיטרציה קודמת. לאחר בחרית האוכלוסייה, כל מועמד שנבחר עבר סדרה של שינויים אקרים: גם המוטציות נבחרות באקרים, וגם המוטציות שנבחרות לא בהכרח יבוצעו, זה תלוי בהסתברות שלהם לביצוע. לאחר ביצוע סדרת השינויים המנגינה המועמדת מסומנת בדגל IsDirty, בכך שהאלגוריתם "ידע" שיש להעיר את טיב האיכות שלא חדש לאור השינוי (מנגנים חדשות שנוצרו באיטרציה הנוכחית בכל מקרה תמיד מסומנות בדגל זה בין אם עברו שינוי ובין אם לאו). בסיום לאחר תום כל השינויים ההסתברות של מוטציות השינוי השונות מופחתת בהתאם לאמור בסעיף סקלבייליות.

מוסכמתות חתימה: מתודות המוטציה הן בעלות מוסכמה הגנרטית הבאה בחתימה שלhn –

`(MelodyCandidate melody, int? barIndex = null)`

הגנרטיות של מוסכמה זו מספקת דינמיות: מתודת ניהול המוטציות אינה צריכה להכיר את המתודות הקונקרטיות כל עוד הן נרשומות בטבלת `Dictionary` של המוטציות `_barMutations`. לפיכך, אם יש צורך במוטציה בעל חתימה אחרת, כדי `יעי` מתודת האתחול `RegisterMutators`. לשמר את הגנרטיות נדרש תבלה נוספת עם חתימה תואמת, לרשות בה את המתודות הרלוונטיות באתחול ולגשת לטבלה זו במתודת `הניהול`, בנוסף לזו הקיימת. החישרונו באחדות הוא שמתודות המוטציה הקיימות צרכות להתיישר: לפיכך אם יש מתודת שינוי שכן פועלת על תיבה ספציפית, ואינדקס מספר התיבה לא סופק לה, עליה להמציא מספר תיבה בעצמה (למשל `יעי` הגרلت מספר אקרים).

מוסכמות שט: מתודות המוצביה הן בעלות מוסכמת השם הבאה : `<Description>Mutation` כאשר `<Description>` מוחלף בתיאור השימוש (שם עצם/פועל) למשל : `.ReverseBarNotesMutation`

תקציר המתוודות במודול: להלן המתוודות השונות המוגדרות במודול זה – מתוודת רישום המוטציות (RegisterMutators), מתוודת ניהול המוטציות (Mutate) והמוטציות עצמן. תיעוד מלא – מפורט בגוף קבצי קוד המקור

Method	Description
ChordPitchMutation	Selects a random note in the requested bar and changes its pitch to one of the chords pitches.
DurationUnifyMutation	Randomly selects two consecutive notes in the given bar, or in a randomly selected bar if no bar index is supplied, and unifies the two consecutive notes into one note by removing the consecutive note entirely and adding it's duration length to the first note.
DurationAnticipationSplitMutation	Replaces a random note in the given bar with two new shorter notes with durations that sum up together to the originals note duration.
DurationDelaySplitMutation	Regarding pitch, one of the new notes after split would have the originals note pitch, and the other note after split would have a pitch which is minor or major second away from the original note pitch.
DurationEqualSplitMutation	Regarding the new durations, they are set according to the requested ratio or a randomly selected ratio.
DurationSplitMutation	Alter the state of candidate solutions.
Mutate	Registers mutation methods with their corresponding default probabilities to operate.
RegisterMutators	Reverses the order of all note sequences in the given melody.
ReverseAllNotesMutation	Reverses the order of the note sequence in the given bar/chord, or in a randomly selected bar/chord. The reverse operation is made in place locally to each chord.
ReverseBarNotesMutation	Selects a random note in the requested bar and changes its pitch to one of the scale pitches.
ReverseChordNotesMutation	Swaps the positions of two chord notes in the given bar.
ScalePitchMutation	Syncopes a bars first note by preceding its start time to its preceding bar, on behalf of the duration of its preceding note (last note from preceding bar).
SwapTwoNotesMutation	Replaces a random hold note with a concrete note pitch and vice-versa.
SyncopedNoteMutation	Replaces a random hold note with a concrete note pitch and vice-versa.
ToggleFromHoldNoteMutation	Replaces a random hold note with a concrete note pitch and vice-versa.
ToggleToHoldNoteMutation	Replaces a random hold note with a concrete note pitch and vice-versa.

חלק גדול מהמוטציות מtabסס על אבני הבניין שכבר הוגדרו במחלקה האב Composer

5.3.3.1 מודול שערוך (FitnessEvaluators.cs)

רקע : סעיף זה מתאר את המודול האחראי על שערוך טיב איכות המנגינות. מודול זה מרוכז בקובץ : FitnessEvaluators.cs. המודול מגדר ומנוהל מתודות האחראיות על שערוך טיב איכות המנגינות השונות שהאלגוריתם מפיק. שערוך זה משתמש כשלב הכנה מקדים לתהליך הסלקציה של האלגוריתם, שקובע אלו מבין כל המנגינות ימשכו לשלב הבא, תוך מתן שיקול משמעותי לדירוג הנינתן להם ע"י שגורות השערוך.

אופן ביצוע השערוך : עברו כל ממד המשמש להערכת מנגינה נתונה, מגדרים מתודות שערוך ייעודית ומשקל יחסית. מתודות השערוך הייעודית אחראית לאמוד את המנגינה ביחס לממד הנתון ולהחזיר מספר מסוימ בין 0 ל-1, המיצג ציון שנמצא ביחס לטיב איכות המנגינה ביחס לממד זה : ככל שהציוון גבוה יותר, הדירוג של המנגינה גבוהה יותר. המשקל היחסית מייצג את החשיבות של הממד לעומת שאר הממדים.

דוגמא – נתבונן במדד בסיסי הבודק עד כמה צלילי המנגינה "רציפים", במובן שהמרוחחים בין זוגות צלילים סמוכים הם ייחסית קתינים (למשל נוכאים מאיזשהו סף קבוע מראש). מתודה השערוך תבצע סריקה לינארית על רצף התווים של המנגינה, תחשב את המרווח בין כל זוג צלילים סמוכים ותמונה כמה מתוכן קפיצות תקינות וכמה חריגות. הציוון שיוחזר מהשערוך יהיה היחס של מספר הקפיצות התקינות מתוך כלל הקפיצות (נ-ט עברו רצף של n תווים). לבסוף ציון זה יוכפל במשקל היחסית שМОוקצת למדד זה. באופן דומה מחושבים ציונים עברו כל יתר הממדים ולבסוף כלל הציונים משוקלם לשכום משוקל שמחווה את הציוון הכללי של המנגינה.

מתודות ניהול השערוך : המתודה EvaluateFitness היא המתודה האחראית לנחל את תהליך השערוך – היא עוברת על הממדים השונים, אומרת אותם ע"י הפעלת פונקציות פונקציית השערוך המתאימות להם תוך התייחסות למשקליהם היחסיים, ולבסוף מסכמת את התוצאות לציוון משוקל. זהה למעשה המתודה היחידה שמשופעת מתוך האלגוריתם הגנטי בהקשר השערוך. כל יתר המתודות נקבעות מתוך המתודה EvaluateFitness.

מחלקה ניהול המשקלים : עברו המשקלים היחסיים, הוגדר סט משתנים קבועים עם ערכי

MelodyEvaluatorsWeights
+ Factor : double
+ ContourDirection : double
+ ContourStability : double
+ DensityBalance : double
+ ExtremeIntervals : double
+ PitchRange : double
+ PitchVariety : double
+ SmoothMovement : double
+ Syncopation : double
+ PitchVariety : double
- _weightSum : double
+ WeightSum : double

ברירות מחדל. לאחר ששימוש במשקלים לשערוך הוא עניין תלוי מימוש שפנימי לוגיקה שמנהלת את השערוך, משתני שערוך אלו פחות שיעיכים למחלקה הראשית של האלגוריתם הגנטי. בכך להימנע מ-"זיהום" המחלקה הראשית, וכן מתן מסגרת ל-DTO שיוכל לרכז נתונים מובוקשים עברו המשקלים משתמשי קצה דרך Web service למשל, הוגדרה מחלקה POCO ייעודית לרכיב המשקלים – המחלקה MelodyEvaluatorsWeights. מחלקה זו מרכזת את כלל הממדים עם משקל ברירת המחדל שלהם, ערך ברירת מחדל לפקטור כללי שנitin להוסיף לכל המועמדים (לכל המנגינות). מחלקה זו מוגדרת גם היא בקובץ FitnessEvaluators.cs.

משמאלי מוצגת דיאגרמת מחלקה שלה. היא פשוט מבנה בדומה ל-C/C++ struct בשפת .

מתודות שערוך טיב איכות המנגינות : מתודות השערוך הן בעלות מוסכמת השם הבא : Evaluate<Criteria>, כאשר <Criteria> מוחלף בשם הממד, למשל : EvaluateDensityBalance. מתודות אלו מפורטות בטבלה בעמוד הבא.

להלן מתודות השערוך השונות ותיאור קצר שליהן. תיעוד מלא מפורט בגוף קבצי קוד המקור –

Method	Description
EvaluateAccentedBeats	Evaluates accented beats in each bar, in terms of the accented pitches and their preceding notes, regarding the transition they create towards the accented beats.
EvaluateContourDirection	Evaluates fitness according to the melody's contour direction. Melodies which tend to have more directional flow, i.e., sequences of ascending and descending notes, would generally score higher.
EvaluateContourStability	Evaluates fitness according to the melody's contour direction stability. This evaluation differs from contour direction by evaluating consecutive sequences of directional intervals, assuring the ups and downs are not randomly distributed, but rather stable consistent.
EvaluateDensityBalance	Evaluates fitness according to the note density balance across the melodies bars. This fitness function objective is to assure the amount of notes in each bar is more or less balanced, and mitigate obscure sounding phrases of which one bar is very dense and another is very sparse, which in general leads to an unpleasant drastic change in feel.
EvaluateExtremeIntervals	Evaluates the ratio of extreme pitch intervals.
EvaluateSmoothMovement	Evaluates how "smooth" the movement is from tone to another.
EvaluateSyncopation	Evaluates fitness according to the amount of existing syncopations. This fitness is calculated as the ratio between the amount of existing syncopations in the melody, and the total amount of real pitched notes, i.e., not hold and rest notes.
EvaluatePitchRange	Evaluates fitness according to the melody's pitch range. This fitness is calculated as the ration between the user's requested pitch range, and the actual candidate's melody pitch range.
EvaluatePitchVariety	Evaluates fitness according to the variety of distinct pitches in use.

5.3.3.1 מודול סלקציה (Selectors.cs)

רקע : סעיף זה מתאר את המודול האחראי על תהליכי הבחירה של המנגינות. מודול זה מרוכז בקובץ : Selectors.cs. המודול אחראי על הגדרת הלוגיקה לבחירת המנגינות שימושים שלב הבא בכל מחזור (دور/איטרציה) של האלגוריתם. ישנו גישות נוספת לבחירת המועמדים שימושיים לשלב הבא, חלקן דטרמיניסטיות וمستמכות אך ורק על הדירוג של המועמדים, ואילו אחרות אי-דטרמיניסטיות משלבות אלמנט של אקריאיות ונותנות צ'אנס גם למועמדים עם דירוג נמוך יותר להמשיך לשלב הבא ולהשתפר בהמשך. כמו כן, חלקן בוחרות רק מועמדים מהדור החדש, חלקן משלבות מועמדים מדורות אחרות. חלק מה希יטות משלבות כמה טכניקות שונות.

שימוש הבחירה במערכת : בשלב זה המערכת מספקת שני שימושים שונים – האחד דטרמיניסטי, והשני אי-דטרמיניסטי –

*** השימוש הדטרמיניסטי (PlusSelection)** שהוגדר עובד בשיטה הנكرة "בחירה פלוס" : חצי מהנקודות משורינים למועמדים "צעירים" שנוצרו בסביב (دور/איטרציה) הנוכחי, ואילו יתר הנקודות מוקצים למועמדים הוותיקים יותר מדורות קודמים. שני הפעולות (צעירים וותיקים) מבוצעת בחירה של המועמדים הקיימים ביותר, דהיינו אלו בעלי הדירוג הגבוה ביותר. להלן שם המתודה .

*** השימוש האי-דטרמיניסטי (RouletteWheelSelection)** שהוגדר, עושה שימוש בمعنى "רולטה מזל" : לכל מועמד מוקצת רצואה על הרולטה, בהתאם ליחס שבין ציון טיב האיכות שלו לסכום הציונים של כלל המועמדים, ואז ע"י הגרלת מספר אקראי בטוח מוגדר ומיפוי המספר שיצא לאחת מהרצאות ע"י מיפויו מראש על הטוחה המוגדר, מבוצע דימוי של סיבוב מהוג מלactivo, ובחר המועמד שהמחוג "עצר" על הרצואה שלו. גם כאן יש עדיפות למועמדים שיש להם יותר, שכן מועמדים שיש להם יותר זכויות לרצואה גדולה יותר וכך מגדים את ההסתברות שלהם לזכיה, אולם משולב גם אלמנט של אקריאיות המקנה גם למועמדים אחרים הזדמנות. היתרונו בהכנסת אקריאיות ומטען הזדמנות גם למועמדים פחות כשרים הוא שזה מסייע בלחימה מהחיפוש להיתקע על נקודות קיצוץ מקומיות, ומגדיל את מרחב החיפוש הרחב, ולא רק עמוק. החישرون הוא שזה כורך בסיכון מסוים וכן זמן ההתכנסות לפתרון איקוני מתארך. להלן שם המתודה :

המתודה **SelectNextGeneration** מספקת מעטפת למשתמשים השונים. זאת המתודה שהאלגוריתם הגנטי קורא לה בבקשת ביצוע את הבחירה/סינון, והיא בתורה אחראית לביצוע שימוש המתאים. נכון לעכשיו מובוצעת קריאה ישירה למוגדרת הבחירה הדטרמיניסטית, אולם ניתן ביכולת להגשים את הבחירה השימוש ע"י הגדרת תלות של מוגדרת הבחירה הראשית (SelectNextGeneration) באיזשהו מנשך או אבסטרקטיה שימושיים אלגוריתם בחירה (אסטרטגייה), ולהזrik מימוש קונקרטי (Dependency Injection) תוך שימוש ב.Factory. אפשרות נוספת היא לשלב בין שיטת הבחירה השונות ולהחליפה ביניהן לשירותים, או קבוע בכל מחזור (دور/איטרציה) את שיטת הבחירה מחדש ע"י "הטלה מטבע" בצדקה של הגרלת מספר אקראי).

להלן סיכום המוגדרות במודול זה. תיעוד מלא שלhn נמצא בגוף קבצי קור המקור –

Method	Description
SelectNextGeneration	Selects a partial set of candidate solutions to form the population of solution candidates for the next generation. The selection process is based on the candidate's fitness (score).
PlusSelection	Filters out the current population of candidates by deterministically selecting only a limited amount of candidates from two separate populations: Current generation candidate's population, and the elder candidate's population.
RouletteWheelSelection	Filters out the current population of candidates with an undeterministic algorithm that simulates a roulette wheel spin, and gives the candidates proportional chances to be selected according to their proportional fitness.

5.3.4 הלחנה עם אלגוריתמים נוספים

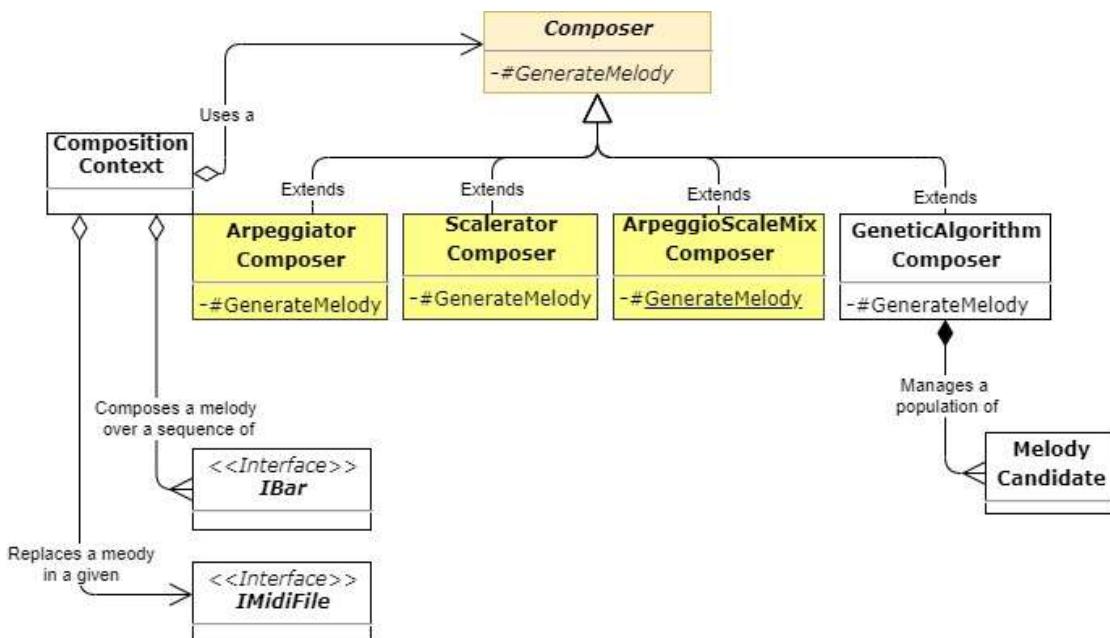
אלגוריתמי הלחנה במערכת מוגדרים במחלקות היורשות ממחלקה האבסטרקטית [Composer](#). [Composer](#) המתווכם ביחסם המתווכם ביותר במערכת ([האלגוריתם הגנטי](#)) מוגדר במחלקה [GeneticAlgorithmComposer](#). בנוסף למימוש אלגוריתם מסויל זה, מוגדרים שלושה אלגוריתמים נוספים, בסיסיים יותר, שפושים מחוללים רצף תווים סטטי המתאים לרקע ההרמוני של היצירה. להלן המחלקות המממשות את האלגוריתמים הבסיסיים הללו ומרחיב השמות שלhn –

#	Class	Namespace
1	ArpeggiatorComposer	CW.Soloist.CompositionService.Composers.Arpeggiator
2	ScalculatorComposer	CW.Soloist.CompositionService.Composers.Scalculator
3	ArpeggioScaleMixComposer	CW.Soloist.CompositionService.Composers.ArpeggioScaleMix

שלושת האלגוריתמים שליל נבדלים בינהם במקור סט התווים שבו הם משתמשים לייצרת רצף התווים. המחלקה הראשונה כוללת ברצף התווים שלה אך ורק תווים השיככים לאקורדים ביצירה, ככלمر מול כל אקורד מתנגנים אך ורק צלילים המרכיבים אותו. המחלקה השנייה בוחרת תווים מתוך סולמות המופיעים מול האקורדים, ש络וב כוללים את צלילי האקורד וצלילים נוספים, ואילו המחלקה השלישית מבצעת איזושה שילוב.

שלושת המחלקות הללו פשטוט דורסות את המתודה האבסטרקטית [GenerateMelody](#) לייצרת מנגינה, ובוגר המתודה הן משתמשות במתודות אתחול גנריות שמוגדרות במחלקה האב בצד, ומפעילות אותן על רצף האקורדים (הפליבק), כך שהאתחול מחולל את רצף התווים ו- "שותל" אותו לצד האקורדים ברצף התיבות שהתקבל בקלט לתיאור המהלך ההרמוני ומהזיר אותו.

הדיגרמה שלhn מדגישה את הקשר של מחלקות אלו בكونטקסט של תת-שכבה זו –



5.3.5 מפעל לייצירת מלוחינים

5.3.5.1 רקע

המערכת תומכת ברגע ארבעה אלגוריתמי הלחנה שונים. בכך לא להיתלות בIMPLEMENTATION ספציפי ולהקנות גמישות בשינויים עתידיים של אלגוריתם הלחנה המבוקש בפתרונות, הוגדר סט קבועים (Enum) שבו מתוארים אלגוריתמי הלחנה השונים שהמערכת תומכת בהם, ומחלוקת מפעל שמקבלת קבוע המיצג אלגוריתם ומהזירה מופע של מחלוקת מלוחין הממשת את אסטרטגיית הלחנה המבוקשת.

המודיבציה מארחורי סט הקבועים ומחלוקת המפעל היא לשבור את התלות של קליניטים בIMPLEMENTATION של אלגוריתם הלחנה כזה או אחר, ולהשאיר תלות רק באסטרטציה, דהיינו רק בחלוקת האב האבסטרקטית Composer. קליניטים המעניינים במופע של אלגוריתם הלחנה יפנו למחלוקת המפעל בבקשת מופע של מלוחין, דהיינו של אלגוריתם הלחנה, ומחלוקת תהיה אחראית לספק מופע שכזה, וכל זאת כדי שהקליניטים צריכים להכיר את [חלוקת שמספקות את המימוש של האלגוריתם](#). זהו בעצם IMPLEMENTATION של עיקרונו – [Dependency Inversion Principle](#).

CompositionStrategy Enumeration 5.3.5.2

הקובץ CompositionStrategy.cs מכל הגדרת Enum עבור סט הקבועים המייצגים את אסטרטגיות הלחנה השונות (דהיינו אלגוריתמים שונים להלחנה) שהמערכת תומכת בהן.

<pre><<Enumeration>> CompositionStrategy</pre>	אוסף זה מאפשר לקליניטים ולחלוקת המפעל הייעודית לתקשר ביניהם בשפה משותפת על אלגוריתמי הלחנה ללא ציון שמות מחלוקת קונקרטיות במפורש.
<pre>ArpeggiatorStrategy ArpeggioScaleMixStrategy ScalculatorStrategy GeneticAlgorithmStrategy</pre>	המערכת תומכת בעת כאמור ארבעה אסטרטגיות השונות. המוסכמה של שמות האסטרטגיות השונות היא "Algorithm Description", כאשר "Algorithm Description" מוחלף בתיאור האלגוריתם. להלן אסטרטגיות הלחנה הנתמכות בשלב זה (모וצגים גם בדיאגרמת – מחלוקת לעיל)

#	Composing Strategy Name
1	ArpeggiatorStrategy
2	ArpeggioScaleMixStrategy
3	ScalculatorStrategy
4	GeneticAlgorithmStrategy

ComposerFactory Class 5.3.5.1

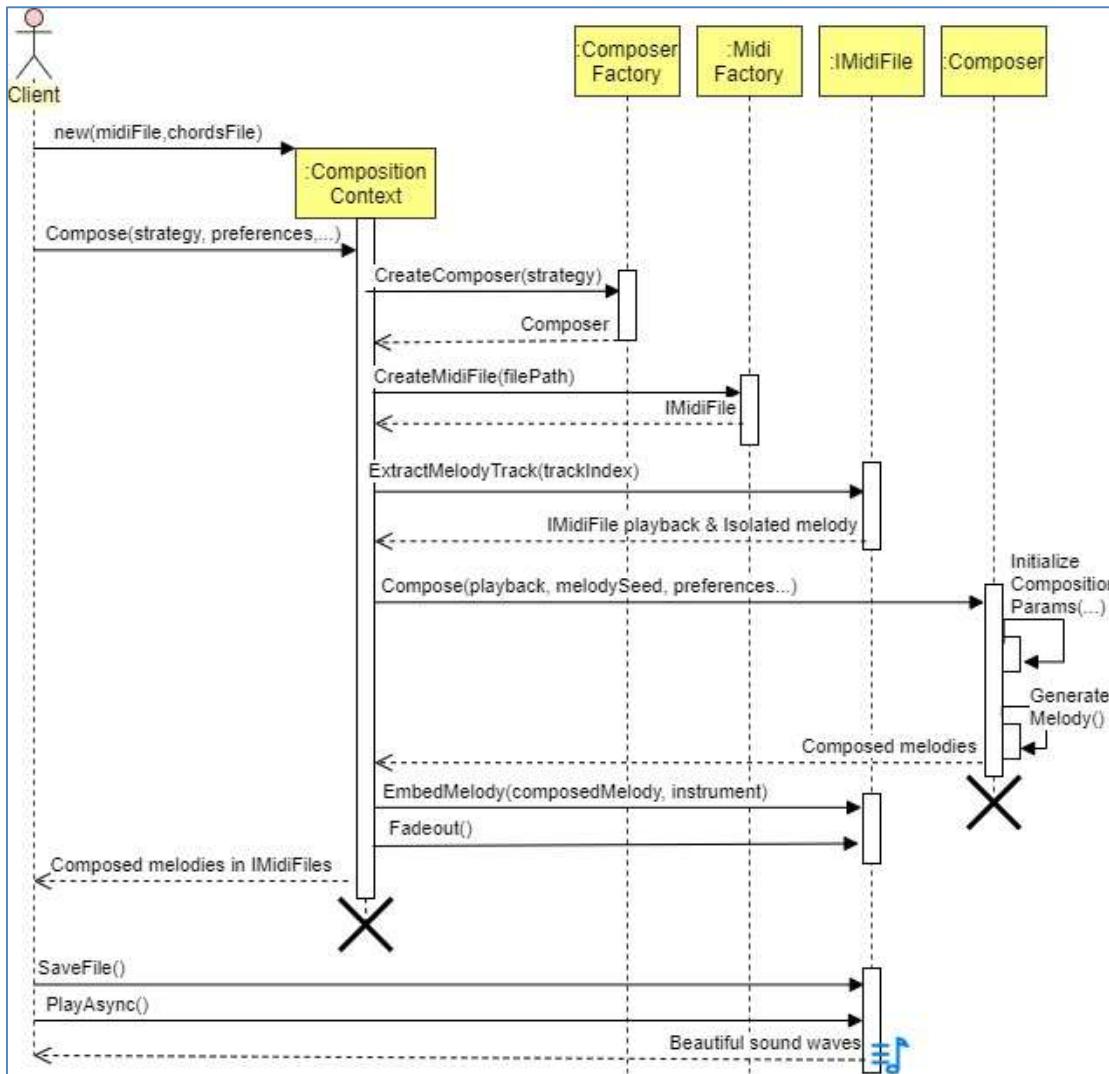
מחלקה ComposerFactory היא מחלוקת סטטית המספקת מетодה STATIC יחידה [CreateComposer](#) (CompositionStrategy). היא מנהלת מיפוי בין אסטרטגיות הלחנה למחלוקת הממשות מהזירה מופע של Composer. היא מפה בין מיפוי אסטרטגיות מול מחלוקת אלו לייצור מופע שלhn, אותן היא מחלוקת קליניטים מבלי שהם מודעים למחלוקת הקונקרטיות המוחזרת. מגנוון זה שובר תלות בIMPLEMENTATION ומאפשר תחזוקה עתידית גמישה בהוספה ועדכון של אלגוריתמי הלחנה וIMPLEMENTATION שלhn.

להלן המיפויים המוגדרים בין אלגוריתמי הלחנה לבין המחלוקות המשותות אוטם, שחלוקת המפעל מהזירה מופע שלhn קליניטים במסגרת המתוודה CreateCompositor בהתאם לאסטרטגיית הלחנה מהם מבקשים –

#	Composing Strategy	Concrete Implementing Class
1	ArpeggiatorStrategy	ArpeggiatorComposer
2	ArpeggioScaleMixStrategy	ScalculatorComposer
3	ScalculatorStrategy	ArpeggioScaleMixStrategy
4	GeneticAlgorithmStrategy	GeneticAlgorithmComposer

5.3.6 אינטראקציה בין היישויות בתהליך הלחנה

להלן דיאגרמת רצף (Sequence diagram) הממחישה את האינטראקציה בין היישויות השונות בתת-שכבה הלחנה ברכז אירועים טיפוסיים שבו קלינינטום מבצעים פניה לשירות הלחנה – (קלינינטום לדוגמה הם אפליקציית Web או Desktop)



תיאור התרחיש בעמוד הבא.

תיאור רצף האירועים בדיאגרמה שלעיל –

1. קליניטים יוצרים מופע של CompositionContext על-סמך קובץ MIDI וקובץ אקורדים של השיר הנדרש וմבקשים מהkontext להלחין מנגינה חדשה עם המתוודה Compose, תוך העברת פרמטרים נוספים כגון אסטרטגיית אלגוריתם הלחנה המבוקשת, העדפות אישיות, אילוצים ואיינדקס רצועת המנגינה בקובץ h-MIDI שהועבר.
2. kontext הלחנה לוקח את המשוכות מכאן ומטפל מול כל יתר הגורמים הרלוונטיים:
 - 2.1. ראשית הוא פונה למפעלים לקבלת מופעים של IMidiFile ו Composer (על סמך האסטרטגייה המבוקשת).
 - 2.2. לאחר מכן הוא מבודד את רצועת המנגינה מתוך קובץ h-MIDI ע"י פניה במתוודה ExtractMelody למופע h-MidiFile, כך שכתע יש בידיו קובץ פלייבק (h-MIDI) לא המנגינה) ואת המנגינה בנפרד, שיכולה לשמש כגרעין השראה לבסס עליו את האתחול.
 - 2.3. עם נתונים מעובדים אלו והפרמטרים הנוספים מהקליניט הקונטקט פונה למלחין בבקשת הלחנה עם המתוודה Compose וממתין לתוצאות.
3. המלחין ראשית מבצע אתחול ולאחר מכן קורא למתוודה הפרטית שלו GenerateMelody שביצעת את תהליך הלחנה בפועל. זהה כאמור מותוודה אבסטרקטית במחלקה Composer שמחקות המלחינים הקליניטים אחרים למש. בסיום העבודה, המתוודה מחזירה לקונטקט הלחנה אוסף של מנגינות.
4. למען פשוטות – התרחיש בדיאגרמה מתאר מצב שבו הקליניט מעוניין במנגינה אחד בלבד מבין אלו שחוירו (למשל זו הראשונה). הקונטקט משਬץ את המנגינה הנבחרת בקובץ MIDI של הפלייבק (הקובץ ללא מנגינה) עם המתוודה EmbedMelody. במידה ויש עניין ביותר מנגינה אחת הקונטקט יוצר פלייבקים נוספים עם המתוודה CreatePlayback של IMidiFile וחזור על התהליך עבור כל אחת מהמנגינות.
5. לבסוף הקונטקט מחליש את הולויים בתיבות האחוריות בקובצי h-MIDI ומחזיר אותם יחד עם המנגינות החדשות שהלחנו בחזרה לקליניט בדמות של מערך של IMidiFile.
6. הקליניט יכול לבקש ממופיע h-MidiFile להתגונן עם המתוודה Play (או PlayAsync) ו/או לשמר עותק פיזי על הדיסק הקשיח של קובץ h-MIDI המיוצר ע"י מופע h-MidiFile.
7. עם קובץ MIDI שמור הקליניט יכול להיעזר בתוכנות צד ג' בשביל להמשיך לעבד את קובץ h-MIDI – להמיר אותו למסמך תווים להדפסה, לנגן אותו, לשנות לו את גובה הצליל, ועוד'.

הרחבות (ComposerExtensions) 5.3.7

מחלקה כזו היא, מגדירה אוסף של Extension Methods שירוטי הלחנה, המרchiebot פונקציונליות עבור טיפוסים של רצפים ואוספים של תווים, דוגמתן – מיון וערבול. מתודות אלו שימשו ליצירת פרמוטציות שונות של תווים במסגרת תהליך הלחנה. להלן המתודות המוגדרות במחלקה זו –

Method	The Subject Entity	Description
Shuffle<T>	IList<T>	Shuffles a sequence of elements of type T in place
Shuffle<T>	IEnumerable<T>	using Durstenfeld's implementation of the Fisher-Yates shuffle algorithm.
Sort	IList<INote>	Sorts a note sequence in ascending/descending order according to the notes' pitches.

אוספי קבועים (Enums) 5.3.8

מחלקה השמות Enumerations CW.Soloist.CompositionService.Enums מגדיר אוסף של המגדירים אוספי עזר של קבועים שנמצאים בשימוש תדיר בתחום הלחנת המנגינות, כגון קטגוריות פרמטרים למגוון המשמעות על אופן הביצוע. כמו כן מוגדרת במרחב שמות זה מחלקת הרוחבות כללית עבור Enums. בסעיף זה מפורטים כל אוספי קבועים אלו ומחלקות החרחות.

EnumExtensions 5.3.8.1

מחלקה זו מגדירה Extension Methods עבור Enums, שטרכן לספק תיאור מפורט מעבר לתיאור המילולי של הקבועים עצמם. יתר על כן, תיאור זה יכול לשמש אחזור אוטומטי של ליישומי Web בפלטפורמת ASP.NET MVC – במסגרת שימוש באנווטציות מתאימות. להלן – המתודות המוגדרות במחלקה זו –

Method	The Subject Entity	Description
GetDescription	Enum	Returns a friendly string representation for the given enum value, which is defined for it in a custom DescriptionAttribute annotation.
GetDisplayName	Enum	Returns a friendly string representation for the given enum value, which is defined for it in a custom DisplayAttribute annotation. This extension is primarily used for supporting the display of the friendly descriptions in ASP.NET MVC applications.

ChordNoteMappingSource 5.3.8.2

אוסף זה מגדיר קבועים המייצגים מקור למיפוי תווים מול אקורד נתון –

Name	Description
Chord	Map the notes from the chord's structure arpeggio notes, for example, map C major chord to notes C (do), E (mi) and G (sol).
Scale	Map the notes from a pre-determined scale that contain notes that are compatible to the chord, a major chord might be mapped to the Dorian Major scale, and dominant 7 chord might be mapped to the blues scale. The actual notes in the mapping scale are implementation dependent.

DurationSplitRatio 5.3.8.3

– אוסף זה מגדיר קבועים המייצגים יחס לפיצול משך שהוא –

Name	Description
Equal	Represents an equal ratio split of 1: 1. Two new durations after the split have an equal duration which is half of the original duration.
Anticipation	Represents an unequal ratio split of 1: 3. The first duration after the split would be $\frac{1}{4}$ long in relation to the original duration, and the second duration would be $\frac{3}{4}$ long in relation to the original duration. For example, if the original duration was $\frac{1}{2}$, then after the split the first duration would be $\frac{1}{8}$, and the second duration would be $\frac{3}{8}$.
Delay	Represents an unequal ratio split of 3: 1. The first duration after the split would be $\frac{3}{4}$ long in relation to the original duration, and the second duration would be $\frac{1}{4}$ long in relation to the original duration. For example, if the original duration was $\frac{1}{2}$, then after the split the first duration would be $\frac{3}{8}$, and the second duration would be $\frac{1}{8}$.

NoteSequenceMode 5.3.8.4

– אוסף זה מגדיר קבועים המייצגים סדר כיוון לרצף תווים נתון ע"פ גובה הצליל שלהם –

Name	Description
Ascending	Notes are aligned in ascending / descending order according to their pitch.
Descending	
ChordZigzag	Notes order is aligned in an alternating order: ascending and then descending or vice-versa. The alternation point is either at a new chord or a new bar.
BarZigzag	

OverallNoteDurationFeel 5.3.8.5

– אוסף זה מגדיר קבועים המייצגים משך שהוא שיינטן למרבית התווים, מה שמשפיע על רמת הדחיסות הכלולת של צלילים במנגינה וקובע את תחושת הקצב הכללית –

Name	Description
Slow	Slow moderate quarter-note based feeling.
Medium	Medium flowing eighth-note based feeling.
Intense	Fast intense sixteenth-note based.
Extreme	Extreme super-fast thirty-second-note based feeling.

Permutation 5.3.8.6

– אוסף זה מגדיר קבועים המייצגים פרמוטציות אפשריות שונות לרצפי תווים –

Name	Description
Shuffled	Notes are aligned in a shuffled random order.
Reversed	Notes original order is reversed.
SortedAscending	Notes are sorted in ascending order according to their pitch.
SortedDescending	Notes are sorted in descending order according to their pitch.

PitchRangeSource 5.3.8.7

אוסף זה מגדיר קבועים המייצגים מקור אפשרי לקביעת מנעד (טוווח) צלילים אפשרי למנגינה, או הטוווח שהמשתמש בעצמו קובע, או הטוווח שקיים למנגינה הקיימת בקובץ ה-MIDI-

Name	Description
Custom	Pitch range would be determined by the user's custom selection.
MidiFile	Pitch range would be determined according to the existing lowest and highest pitches in the existing melody in the midi file.

SortOrder 5.3.8.8

אוסף זה מגדיר קבועים המייצגים סדר מיון אפשרי (עליה/ירידה) –

Name	Description
Ascending	Ascending order sequence. Elements are sorted in increasing order from small to big.
Descending	Descending order sequence. Elements are sorted in decreasing order from big to small.

6 שכבת הגישה לנתונים (DAL)

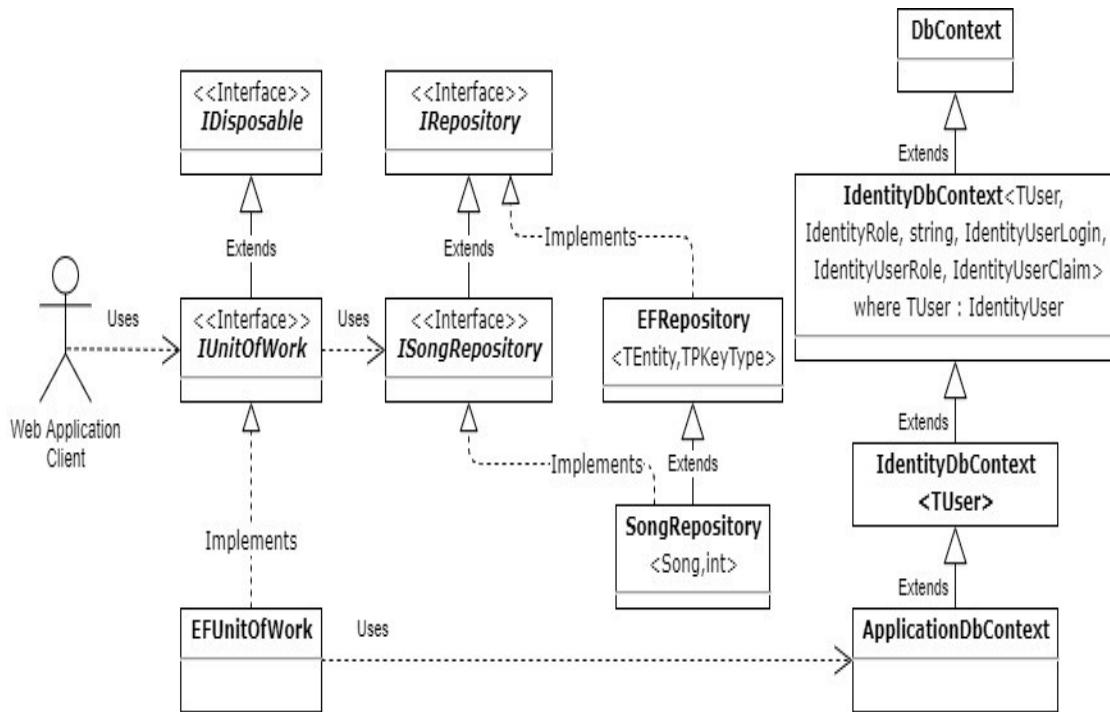
שכבה זו של ה-(DAL) Data Access Layer מטפלת בכל הלוגיקה הנדרשת לגישה לנתונים – חיבור המערכת למסד נתונים פיזי, ניהול הגישה למסד נתונים זה מתוך המערכת, אחזור ועടכון של הנתונים, המרת הנתונים מאופן ייצוגם במדיום האחסון הפיזי לאופן ייצוגם התואם במערכת בעולם האובייקטיבים של התכונות מונחה עצמים (ולהיפך), ואספקת אבטרכציות ושירותי גישה עוטפים לקליינטים לכך שיוכלו לגשת לנתונים על סמך חזה התקשרות מוסכם ללא תלות באופן המימוש הפיזי של הגישה ולא תלות במדיום הפיזי שבו הנתונים נשמרים בפועל מחורי הקלעים.

השכבה מוגדרת בשלמותה כספרייה בפרויקט ייעודי המהווה יחידת אסambilי עצמאית: DataAccess (ווע"פ [מוסכמות השמות במערכת למרחבי שמות](#)), כל מרחבי השמות namespaces (namespaces) השיביים לשכבה זו מושרים תחת מרחב השמות: CW.Soloist.DataAccess, המחולק לרכיבים לוגיים שונים תחת מרחבי שמות נפרדים, כאשר כל רכיב אחראי על השלמת פונקציונליות/שירותים מסוימים. להלן סקירה קצרה של הרכיבים השונים את שכבת הגישה לנתונים עם מרחב-השמות הפנימיים שלהם (ב العمודים הבאים רכיבים אלו מותוארים בפירוט בחלוקת לפי מרחבי השמות הבאים) –

# נתונים	תפקיד תת-רכיב בשכבת הגישה	שם מרחב שמות (Namespace) של הרכיב
1	ספקת שירות ניהול גישה לנתונים לקליינטים באמצעות הגדרת אבטרכציה (ומימוש) של מנהל ישוות המתפקיד כ-Unit of Work אשר אחראי על מעקבי שינויים ו互動ראקציה בין הרכיבים השונים בשכבה זו.	CW.Soloist.DataAccess.UnitOfWork
2	רכיב כל המודלים – הישויות העסקיות במערכת הרלוונטיות לאחסון (Persistence).	CW.Soloist.DataAccess.DomainModels
3	הגדרת אבטרכיאות לגישור בין המודלים (ישויות עסקיות הרלוונטיות Persistence) לבין מדיום האחסון הנתונים הפיזי, ורכיב מימושים של אבטרכיאות אלו.	CW.Soloist.DataAccess.Repositories
4	רכיב קונפיגורציות וيسויות המספקות שירותים ORM – שירותים מייפוי בין אובייקטיבים לרשומות בסיסי נתונים רלוונטיים באמצעות שירותים הספרייה EntityFramework של EntityFramework.	CW.Soloist.DataAccess.EntityFramework
5	רכיב מיגרציות (עדכונים) על הישויות ובasis הנתונים שנמצאים בפיקוח של EntityFramework במטרה להביא אותן לישור קוו בתאיימות גרסאות עם יכולת Rollback.	CW.Soloist.DataAccess.Migrations

מהצד של הקליינטים של שכבת הגישה לנתונים – שכן למעשה שכבות אחרות במערכת העושות שימוש בשכבה זו, דוגמת צד השירות בשכבת התצוגה באפליקציה ה-Web-ית בתת-שכבת שירות (Service Layer), הקליינטים מכירים רק את המודלים, שכן מחלקות ה-POCO של הישויות העסקיות בעולם הקונספטואלי, ואת מנהל הישויות במנשך IUnitOfWork. כל המימוש הפנימי באמצעות Entity Framework שקוף לклиיניטים. הם מנהלים את כל האינטראקציה מול מסד הנתונים באמצעות מופע של IUnitOfWork שמשמש כ-Database gateway. הוא מנהל את היחסות השונות הרלוונטיות ל-Persistence וכל הבקשות לאינטראקציה מול ה-DB בעברות אליו ומטופלות על ידו.

להלן דיאגרמת מחלקה המתארת את הקשרים בין הרכיבים המרכזיים בשכבה הגישה לנוטונים –



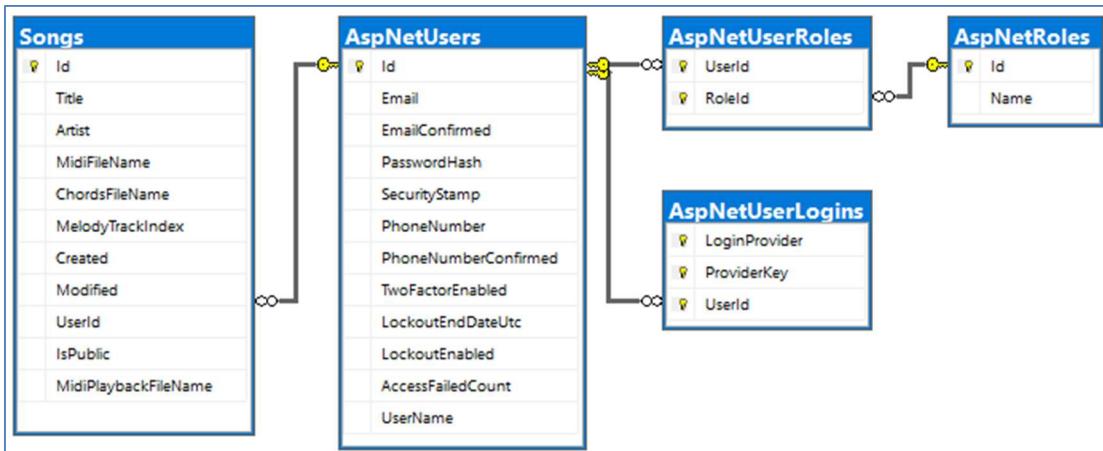
כפי שצוין לעיל, היישום שמשתמש בשכבה הגישה לנוטונים מנהל אינטראקציה אך ורק עם מנשך `IUnitOfWork`, כך שגם בעתיד ידרש להחיליך את המימוש הפנימי של שכבה זו, כל עוד שהמימוש החדש ימשיך לעמוד בחוזה הכללי שמוגדר ב-`IUnitOfWork`-ב-`Entity` Framework או פלטפורמה ספציפית אחרת, ה החלפה תהיה שקופה לחילוטן לקליינטים, למעט אולי עדכון חד-פעמי של רישום התלוויות חד-פעמיות בكونפיגורציה של ה-`IoC` להיפוך והזרקת התלוויות (Inversion Of Control) המוגדרת באמצעות ספריית `AutoFac`.

6.1 מסד הנתונים (Database)

המערכת מארחנת את הנתונים שמשמשים את האפליקציה בסיס נתונים רלוונטי. בסעיף זה מפורט מבנה בסיס נתונים זה.

6.1.1 דיאגרמת טבלאות

להלן דיאגרמה המתארת את הקשרים בין הטעלות השונות בסיס הנתונים –



6.1.2 פירוט הטעלות

להלן תקציר הטעלות המרכזיות בסיס הנתונים –

Table Name	Content
Songs	Songs uploaded to the application.
AspNetUsers	Registered users of the application.
AspNetUserLogins	External third-party user account of application registered users.
AspNetRoles	Authorization roles.
AspNetUserRoles	Assignment of roles to registers users.

בתת-סעיפים הבאים מפורטוות הסכמאות של כל אחת מטעלות אלו.

AspNetUserLogins Table 6.1.2.1

טבלה זו מכילה פרטי חשבון חיצוניים של אפליקציית צד גי של משתמשים רשומים, דוגמת Microsoft, Google, LinkedIn או אחרים. חשבון חיצוני יכול לאמת את המשתמש אל מול היישומים החיצוניים באמצעות פרוטוקול הזדהות כגון OAuth. להלן שדות הטבלה –

Field Name	Field Type
LoginProvider	nvarchar(128)
ProviderKey	nvarchar(128)
UserId	nvarchar(128)

AspNetRoles Table 6.1.2.2

טבלה זו מכילה את הרולים (תפקידים) במערכת המשמשים לניהול הרשאות משתמשים רשומים. מוגדרים שני תפקידים – תפקיד Admin של מנהלו מערכת, ותפקיד ApplicationUser עבור כל יתר המשתמשים הרשומים –

Field Name	Field Type	Description
<u>Id</u>	nvarchar(128)	Internal id of the role in the database.
Name	nvarchar(256)	Role name.

AspNetUsers Table 6.1.2.3

טבלת זו מכילה את המשתמשים הרשומים במערכת, את פרטי החשבון שלהם כגון –
המשמשים להזדהות בחיבור כגון כתובת אי-מייל ו-Hash של הסיסמה –

Field Name	Field Type
<u>Id</u>	nvarchar(128)
Email	nvarchar(256)
EmailConfirmed	bit
PasswordHash	nvarchar(MAX)
SecurityStamp	nvarchar(MAX)
PhoneNumber	nvarchar(MAX)
PhoneNumberConfirmed	bit
TwoFactorEnabled	bit
LockoutEndDateUtc	datetime
LockoutEnabled	bit
AccessFailedCount	int
UserName	nvarchar(256)

AspNetUserRoles Table 6.1.2.4

טבלת זו היא טבלת קשר הממחה בין משתמשים לבין הרוליהם המוקצים להם -

Field Name	Field Type
<u>UserId</u>	nvarchar(128)
<u>RoleId</u>	nvarchar(128)

Songs Table 6.1.2.5

טבלת השירים מכילה את נתונים כל השירים שימושי האפליקציה (לרבוט מנהלו המערכת) העלו למערכת. השירים עצם (זהיינו קבצי Midi וקובצי האקורדים של השירים) אינם מאוחסנים בסיס הנתונים עצמו בטבלה ישירות, אלא על שרת הקבצים (File server). בטבלת השירים נשמרים רק הנתיבים לקובצי השירים על שרת הקבצים.

Field Name	Field Type	Description
<u>Id</u>	int	Internal id of the song in the database.
Title	nvarchar(50)	Song name.
Artist	nvarchar(50)	Songs artist (writer/band) name.
MidiFileName	nvarchar(100)	Path on file server to songs midi file.
ChordFileName	nvarchar(100)	Path on file server to songs chord file.
MelodyTrackIndex	int	Index in the midi file of the melody track that is to be replaced when composing new melodies.
Created	datetime	Timestamp of song upload.
Modified	datetime	Timestamp of last song update.
<u>UserId</u>	nvarchar(128)	Id of the registered user who uploaded the song.
IsPublic	bit	Flag indicating whether the song is visible to all users (public) or only to the user who uploaded the song.
MidiPlayBackFileName	nvarchar(100)	Path on file server to songs midi playback file (midi file without the melody track).

6.2 מודלים (Domain Models)

מודלים (Domain Models) הם מחלקות עבור ישויות עסקיות במערכת שרלוונטיות ל-Persistence, דהיינו מחלקות לאחסון בסיס נתונים). מחלקות אלו הן מחלקות POCO (Plain Old CLR Objects), דהיינו מחלקות פשוטות שלא נדרש מהן פונקציונליות מיוחדת שמיוחסת בירושה או מיומש במקרה אחר, אלא רק הגדרת סט מאפיינים של הישות ושגרות בסיסית לעדכון ואחזור מאפיינים אלו. המודלים מספקים ייצוג בעולם האובייקטיבים לטבלאות השטוחות הקיימות בסיס הנתונים – בד"כ מופע של מודל מייצג איזושהי רשות בטבלה בסיס הנתונים שמומפה מול אותו מודל.

בנוסף למאפיינים שמודדים במחלקות מודלים אלו נגד השודט בטבלאות ה-Database, המודלים מגדירים גם מאפיינים לנויוט (Navigation Properties) בין ישויות הקשורות נגד הקשרים שבין הטבלאות. למשל – עבור קשר של רבים-אחד מטבלת השירותים אל טבלת השירותים, המוצג בסיס הנתונים ע"י אילוץ מפתח זר של מזזה המשמש ברשומות של השירותים, במחלקת המודל המייצגת שירותים יהיו שני מאפיינים עבור הקשר לשימוש – מאפיין אחד עבור מזזה המשמש שהעליה את השירות כנגד העמודה התואמת בטבלה ב-Database, ומאפיין שני עבור מופע תואם של אותו משתמש המחלקה שמייצגת משתמשים. באופן דומה הצד השני של הקשר, יהיה מוגדר מאפיין ניוט של רשימה שמכיל את רשימת כל המופעים של שירותים שהועלו ע"י אותו משתמש.

כל המודלים במערכת מוגדרים למרחב השמות הייעודי הבא –

CW.Soloist.DataAccess.DomainModels

סעיף זה מפרט את מחלקות המודלים השונות.

6.2.1 מודל משתמש (ApplicationUser Model)

מחלקה ApplicationUser מייצגת משתמש המערכת. מחלקה זו יורשת מהמחלקה IdentityUser הקיימת ב-.NET, שמכילה פונקציונליות מובנית לתמיכה ב-AAA – Authentication, Authorization & Accounting המתאימים להן בסיס הנתונים – AspNetUsers, AspNetLogins ועוד.

מעבר להגדרות המובנות של .NET, נוספת למחלקה זו הגדרת מאפיין ניוט עבור הקשר שבין משתמשים לשירותים. בכל מופע של משתמש, מוגדרת רשימת כל השירותים שהוא ה耷לה למערכת –

```
public IList<Song> Songs { get; set; }
```

6.2.1.1 שמות רולים (RoleName)

מחלקה הסטטית RoleName מגדירה קבועים כנגד הרולים במערכת –

```
public static class RoleName
{
    public const string Admin = "Admin"; // maximum privileges
    public const string ApplicationUser = "ApplicationUser"; //restricted privileges
}
```

מחלקה סטטית זו מוגדרת בקובץ ApplicationUser.cs יחד עם המחלקה ApplicationUser. לאחר שהוא נוגע לשירותים למאפיינים של מודל המשתמש.

AuthorizationActivity Enumeration 6.2.1.2

אוסף הקבועים AuthorizationActivity (Enum) מגדר סוג פעילות הרלוונטיים למידור ובדיקת הרשותות. האוסף מוגדר בקובץ ApplicationUser.cs יחד עם המחלקה ApplicationUser, מאחר שהוא נוגע ישירות למאפיינים של מודל משתמש. הקבועים המוגדרים הם כדלקמן –

```
public enum AuthorizationActivity {
    Create = 1, Update = 2,
    Display = 3, Cancel = 4, Delete = 5 }
```

מודל שיר (Song Domain Model) 6.2.2

המחלקה Song מייצגת ישות/רשומה של שיר, כנגד הטליה Song בסיס הנתונים.
להלן רשימת המאפיינים המוגדרים במחלקה זו –

Property	Type	Description
Id	int	Id of the song in the database.
Title	string	Song name.
Artist	string	Songs artist (writer/band) name.
Midi	IMidiFile	Midi file content from the midi file.
MelodyTrackIndex	MelodyTrackIndex?	Melody track index in midi file.
Chords	IEnumerable<IChord>	Chord progression of chords described in the chords file.
MidiFileName	string	Path on file server to midi file.
MidiPlaybackFileName	string	Path on file server to chord file.
ChordsFileName	string	Path on file server to songs midi playback file (midi file without the melody track).
Created	DateTime	Timestamp of song upload.
Modified	DateTime	Timestamp of last song update.
IsPublic	bool	Flag indicating whether the song is visible to all users (public) or only to the user who uploaded the song.
User	ApplicationUser	User entity instance of the user who uploaded the song.
UserId	string	Id of the registered user who uploaded the song.

בנוסף למאפיינים שלעיל, המחלקה מגדרת מתודה עבור קביעת שם לקובץ הפליביק שהמערכת מייצרת לאחר העלאת שיר. המתודה אחראית על עדכון המאפיין MidiPlaybackFileName. MidiPlaybackFileName – הגדרת המתודה

```
public void SetPlaybackName(string referenceName = null) { ... }
```

SongFileType Enumeration 6.2.2.1

אוסף הקבועים SongFileType (Enum) מגדר סוגים קבועים של管理 – קובץ אקורדים – קובץ טקסט, קובץ MIDI של השיר המקורי כפי שהועלה למערכת וקובץ פלייבק – קובץ-h-MIDI לאחר שהוסרה ממנו רצועת המנגינה. בהתאם, ערכיהם אלו הוגדרו כדלקמן –

```
public enum SongFileType {ChordProgressionFile, MidiOriginalFile, MidiPlaybackFile}
```

אוסף זה מוגדר בקובץ Song.cs יחד עם המחלקה Song, מאחר שהאוסף נוגע ישירות למאפיינים של מודל השיר.

6.3 מאגרים (Repositories)

כל רכיבי הפיתוח בסעיף זה מוגדרים למרחב השמות הבא –
CW.Soloist.DataAccess.Repositories

קבץ קוד המקור המכילים רכיבים אלו נגישים בקישור הבא –
<https://github.com/cwelt/Soloist/tree/master/DataAccess/Repositories>

מאגר ([Repository](#)) הוא דפוס עיצוב ארכיטקטוני שմガשר בין ישויות עסקיות (מודלים) לבין המיפוי שלהם מול בסיס הנתונים הפיזי, מעין קופסה שחורה שוטפת את הלוגיקה של המיפוי, המספקת לקליניטים יכולת לבצע אינטראקציה מול מדדים-האחסון באופן מופשט, High-level, בטרמינולוגיה של הישויות העסקיות בעולם האובייקטים, מבלתי הצורך להתעסק או לדעת כיצד הישויות מאוחסנים בפועל (טבלאות, עמודות וכו').

המאגר Repository כשמו כן הוא, מאגר. כזה, הוא מנהל אוסף של ישויות במאגר, לרוב מאות טיפוס, ע"י הגדרת פונקציונליות לעובדה מול המאגר – הוספה וגריפה של ישויות אל/מ המאגר, ומילוי המאגר מותך מסד הנתונים.

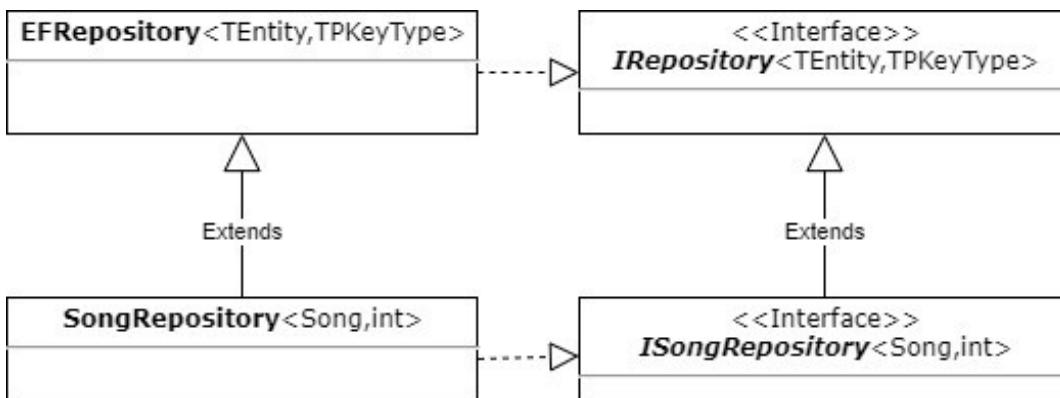
לאחר שמירת שינויים שבוצעו על ישויות במאגר בעולם האובייקטים, השינויים התואמים עבורים לביצוע בפועל במסד הנתונים – ישויות שנגרכו מהמאגר גורמות למחיקת רשומות מהטבלאות התואמות בסיס הנתונים, ישויות חדשות שנוסףו למאגר נוספות כרשומות חדשות טבלאות וכן הלאה.

שם כך ה-Repositories מגדירים פעולות מיילן – CRUD – Create, Read, Update, Delete – ואילו ישות לפי מזחה חד-ערכי (מפתח) או אחזר אוסף כל הישויות העונთ ופעולות כגון – אחזר ישות לפי מזחה חד-ערכי (מפתח) או אחזר אוסף כל הישויות העונთ לסט קרייטריונים. ככל הפעולות האלה מוגדרות ב-Repositories כمتודות, ובಗוף המימוש של מетодות אלו מבוצעת כל הלוגיקה של ה-Low-level של תרגום הקרייטריונים ומאפייניהם מעולם אובייקטיבים לטרמינולוגיה וייצוג בעולם בסיסי הנתונים הרלוונטיים – ה-Repository דואג לתרגם את המethodות לשאלות מתאימות ב-SQL, ולאחר ביצוע שלהם בסיס הנתונים, מעדכן את המאגר עם מופיע הישות בזמן ריצה בהתאם לסתוטוס העדכני שלהם בסיס הנתונים.

מנקודת המבט של הקליניטים שעובדים עם ה-Repositories, הם רק צריכים לקרוא למетодות על הישויות העסקיות, כל העבודה הסבוכה שליל מול בסיס הנתונים מוסתרת ונחסכת מהם.

הטמעת דפוס ה-Repositories במערכת מבוצעת ע"י הגדרת [יעודי](#) נגד כל ישות עסקית (מודול). אם למשל יש טבלת שירים במסד הנתונים בשם Songs וישות עסקית של מודול המיצג שיר במחלקה הנקראת Song, נגיד [יעודי](#), למשל SongRepository, שייהי אחראי על אינטראקציה מול בסיס הנתונים בהקשר של שירים. אם נרצה למשל לשמר בmseד הנתונים גם נתוני פרופיל משתמש למשל, נקים טבלה מתאימה ב-Database, נגיד [UserProfile](#) מחלוקת מודול כנראה, וזה נגיד [UserProfile](#) (למשל UserProfileRepository) שייהי אחראי על אינטראקציה מול ישויות נתוני פרופיל משתמש.

להלן דיאגרמת מחלוקת המתארת את ישויות ה-Repository השונות המרכיבות מודול זה של המאגרים, ואת הקשריהם שביניהם –



6.3.1 מנגש מאגר (IRepository Interface)

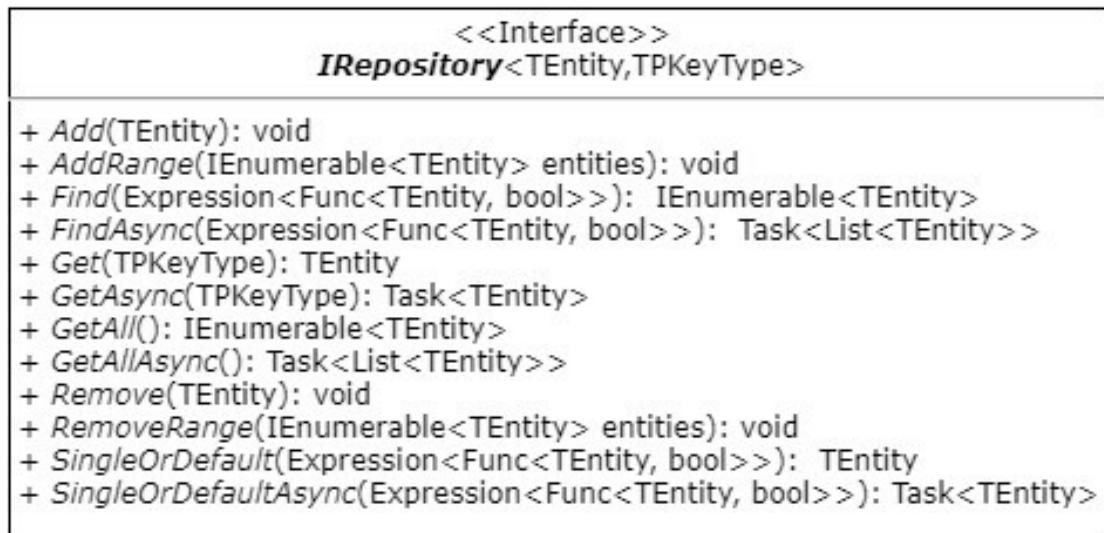
המנגש IRepository מגדיר מנגש-אב גנרי עבור כל מנשיки ה-Repository

```
public interface IRepository< TEntity, TPKeyType > where TEntity : class { ... }
```

ב-Repositories הקונקרטיים, הפרמטר הגנרי `TEntity` מוחלף בטיפוס הישות העסקית של אותו Repository – טיפוס מחלקת המודול הרלוונטי, והפרמטר הגנרי `TPKeyType` מוחלף בטיפוס המאפיין/שדה באותה מחלקת מודול שמייצג את שדה המזהה החד-ערכי (מפתח) של הישות העסקית.

הערה : מימוש זה של טיפוס מפתח מתאים בעיקר עבור מודלים פשוטים שבהם המזהה החד-ערכי (מפתח) אכן מיוצג ע"י שדה יחיד, וזהו אכן המצב במערכות. אם בעtid יידרש להגדיר יצוג תואם גם עבור ישות שהזהוי החד-ערכי שלה מורכב מ-2 שדות או יותר (ולא שדה יחיד), עדין ניתן יהיה לעשות שימוש בתצורה הנוכחית ע"י שרשור השדות השונים המרכיבים את המפתח במחלקת המודול של ישות עסקית זו לשדה ייעודי חדש, `String`, בטיפוס `String`, שמייצג את המפתח בשדה בודד, ואז ב-Repository לפרק שדה זה בחזרה לשדות המרכיבים, ואיתם לגשת לטבלאות בסיס הנתונים. מימוש גנרי יותר יספק תמכה בקבלה אוסף עמודות המייצגות את המפתח, דוגמת מערכ ה-ADO.NET שאובייקט `DataTable` מקבל כפרמטר למפתח בגישה ה-Disconnected Layer.

להלן דיאגרמת מחלקה עבור המנגש IRepository המתארת את מתודות המנגש –



להלן תקציר המתודות המוגדרות במנגש האב-הגנרי IRepository. מתודות אחזור וחיפוש –

Method	Description
Get	Gets the entity that is identified by the given id, either synchronously or asynchronously.
GetAll	Get all entities from this repository either synchronously or asynchronously.
GetAllAsync	
Find	Gets all entities from this repository which satisfy a given condition, either synchronously or asynchronously.
FindAsync	
SingleOrDefault	Returns the only entity from this repository that satisfies a specified condition or a default value if no such entity exists,
SingleOrDefaultAsync	either synchronously or asynchronously.

– מетодות הוספה ומחיקה של יישוות מהמאנר –

Method	Description
Add	Adds the given entity to this repository.
AddRange	Adds the given sequence of entities to this repository.
Remove	Removes the given entity from this repository.
RemoveRange	Removes all the entities in the given sequence from this repository.

עבור כל ישות קונקרטית שRELATIONAL Repository, יוגדר יירוש ISongRepository. Persistence-האב הגנרי עם טיפוסים קונקרטיים. המנשכים הייעודים יכולים להעיר את הממשק הבסיסי בפונקציונליות פרטנית יותר, למשל, מאגר שירים עשוי להגדיר מетодה שמאפשרת לאחזר שירים ע"פ שם המחבר/להקה, או אחזר אוסף כל השירים מסווגים תחת סגנון מוסיקלי מסוים.

קוד המקור של הממשק מוגדר בקובץ IRepository.cs שנייתן למצוא בקישור הבא –
<https://github.com/cwelt/Soloist/blob/master/DataAccess.Repositories/IRepository.cs>

6.3.2 ממשק מאגר שירים (ISongRepository Interface)

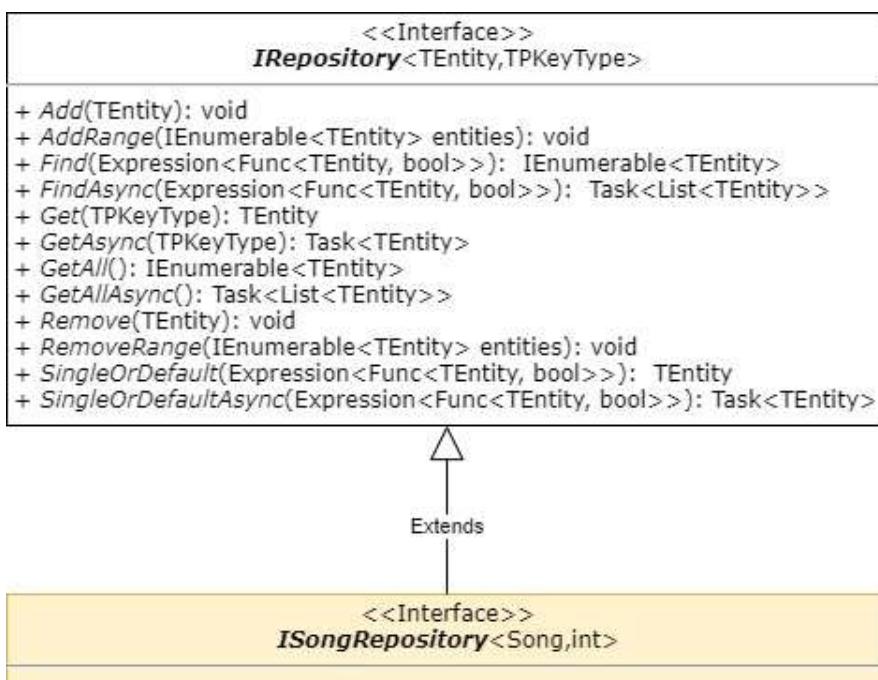
הממשק ISongRepository ירוש מממשק-האב הגנרי IRepository ומגדיר ממשק מאגר עבור יישוות של שירים, המציגים ע"י מחלקה המודול – Song

```
public interface ISongRepostiory : IRepository<Song, int> { }
```

לפי שעה ממשק זה משמש כמנشك תיוג, המגדיר את הטיפוסים הקונקרטיים של הישות העסקית שמאגר זה אחראי עליה – Song, ושל המפתח – המזהה החז-ערך של ישות זו – int. אם יידרשו מетодות למניפולציה על מאגר השירים, כגון אחזר על השירים לפי סגנון מוסיקלי מסוים, הן יוגדרו במנשך זה.

קוד המקור של הממשק מוגדר בקובץ IRepository.cs שנייתן למצוא בקישור הבא –
<https://github.com/cwelt/Soloist/blob/master/DataAccess.Repositories/ISongRepostiory.cs>

להלן דיאגרמת מחלקה עבור הממשק ISongRepository



6.3.3 מחלקה-אב גנריית למימוש מנשך המאגר הגרפי (EFRepository Class)

: [IRepository](#) מגדירה מחלקה-אב גנריית המממשת את מנשך-האב הגנרי

```
public class EFRepository< TEntity, TPKeyType > : IRepository< TEntity, TPKeyType >
where TEntity : class { ... }
```

.EF בשם המחלקה הם ראשי-תיבות של Microsoft [ORM](#), [Entity Framework](#) מבית חבילת [Entity Framework](#) של האופן שבו היא מימושת את המנשך `IRepository` : באמצעות הפונקציונליות שכבר מסופקת "Out of the Box" ע"י חבילת EF. החבילת מגדירה טיפוס `DbContext` המתפרק כמנהל ישותות תוך מימוש הדפוס הארכיטקטוני [Unit of Work](#), ומחזק אוסף ישותות גנריות מטיפוס `DbSet`, אשר כל אחת מהן מתפרקת למעשה כ-Repository עבור הישות מהטיפוס הקונקרטי של אותו `DbSet`. לפיכך, במקרה לכתוב מאפס את כל הלוגיקה למיפוי ישות עסקית מול טבלת DB, המרת מתודות לשאלות SQL וכו', ניתן בכך למימוש מתודות המנשך פשוט לבצע האצלת אחריות (delegation) אל ישותות EF.

היתרון בשימוש בממשק ובמחלקה מגשת זו במקומות שישיר ב-EF בקוד בצד הקליניינטים (יותר שכבות המערכת) הוא האבסטרקציה : הקליניינטים אינם תלויים ב-EF אלא רק בממשק המאפשרת של `IRepository`. התלות באבסטרקציה מאפשרת להחליף את המימוש הקיים של מחלקה זו במחלקה אחרת, שممמשת את המנשך ע"י חבילת ORM אחרת (למשל [NHibernate](#)), או בכלל בשימוש ישיר של ספריית EF-.NET. כל עוד המימוש הוא של המנשך `IRepository`, תוך שימוש ב-Factory מתאים (או מנגנון [Dependency Injection](#)), קליניינטים שעובדים מול המנשך בלבד המחלקה הקונקרטית שممמשת את המנשך שקופה לחולטיין.

מחלקה זו היא כאמור גנריית. עבור היישויות השונות המעוניינות לעשות שימוש במחלקה זו למשוך מאגר יש להגדיר מחלקות קונקרטיות היורשות ממחלקה זו, ולהחליף את הטיפוס הגנריים בטיפוס הקונקרטיים של המודול הרלוונטי וטיפוס המאפיין שמשמש כمزזה החד-ערבי של המודול.

להלן המאפיינים/שדות המוגדרים במחלקה זו : מאפיין `DbContext` שאחראי כאמור על ניהול היישויות ואיינטראקציה מול DB, ושדה `DbSet< TEntity >` שמחלקות מאגר קונקרטיות צרכות להחליף עם הטיפוס של ישות המודול הקונקרטית –

```
protected DbContext Context { get; }
private readonly DbSet<TEntity> _entities;
```

המחלקה מדירה בנאי ייחד שדרכו ניתן "להזיריק" תלויות במאפיינים שלעיל כארגומנטים –

```
public EFRepository(DbContext context)
{
    Context = context;
    _entities = context.Set<TEntity>();
```

להלן דוגמה לאופן ביצוע האצלת הסמכות (delegation) למימוש הקאים ב-EF : המתודה `Get` – מוגדרת במנשך `IRepository` ווברת אדפטציה למетодה `Find` של ישות ה-

```
public TEntity Get(TPKeyType id) => _entities.Find(id);
```

קוד המקורי המלא של המחלקה מוגדר בקובץ `EFRepository.cs` שנייה למצוא בקישור הבא –
<https://github.com/cwelt/Soloist/blob/master/DataAccess.Repositories/EFRepository.cs>

להלן דיאגרמת מחלקה עבור המחלקה `EFRepository` המתארת את רכיבי המחלקה –

EFRepository<TEntity,TPKeyType>

```
# Context: DbContext
- _entities: DbSet<TEntity>

+ EFRepository(DbContext) : EFRepository
+ Add(TEntity): void
+ AddRange(IEnumerable<TEntity> entities): void
+ Find(Expression<Func<TEntity, bool>>): IEnumerable<TEntity>
+ FindAsync(Expression<Func<TEntity, bool>>): Task<List<TEntity>>
+ Get(TPKeyType): TEntity
+ GetAsync(TPKeyType): Task<TEntity>
+ GetAll(): IEnumerable<TEntity>
+ GetAllAsync(): Task<List<TEntity>>
+ Remove(TEntity): void
+ RemoveRange(IEnumerable<TEntity> entities): void
+ SingleOrDefault(Expression<Func<TEntity, bool>>): TEntity
+ SingleOrDefaultAsync(Expression<Func<TEntity, bool>>): Task<TEntity>
```

6.3.4 מחלקה מאגר שירים (SongRepository Class)

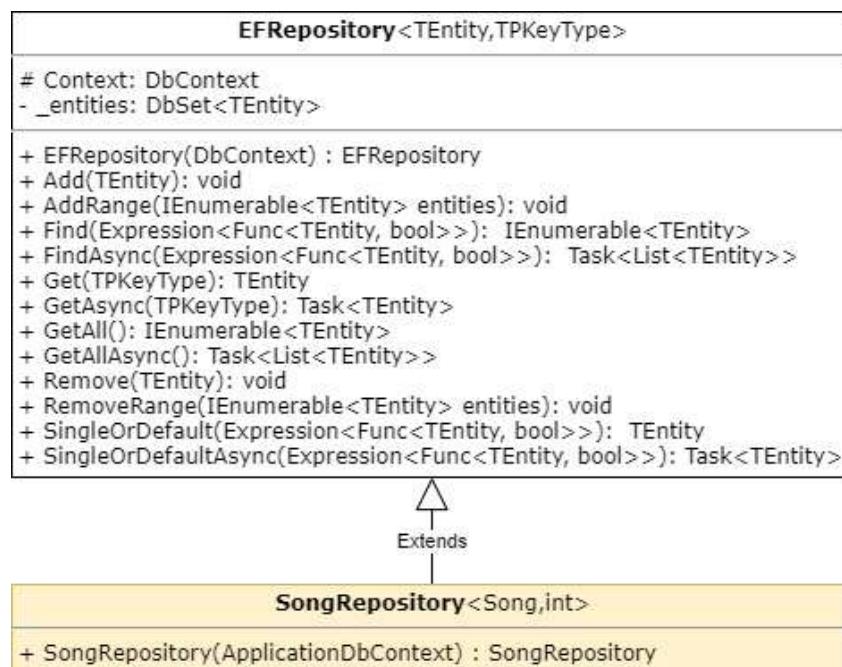
המחלקה `SongRepository` יורשת ממחלקה-האב `EFRepository` ומגדירה מחלקה מאגר עבור ישוות של שירים, המיוצגים ע"י מחלקה המודול – `Song`

```
public class SongRepository : EFRepository<Song, int>, ISongRepostiory
```

לפי שעה מנשך מחלקה זו אינה מרחיבה את הפונקציונליות הקיימת במחלקה האב, אלא רק מגדרה את הטיפוסים הקונקרטיים במקום אלו הגנריים.

קוד המקורי של המחלקה מוגדר בקובץ `SongRepository.cs` שנייה למצוא בקישור הבא –
<https://github.com/cwelt/Soloist/blob/master/DataAccess.Repositories/SongRepository.cs>

להלן דיאגרמת מחלקה עבור המתארת את רכיבי המחלקה וקשר הירושה שלה למחלקות-האב הగנריות –EFRepository



6.4 ניהול יחידות עבודה (Unit Of Work)

כל רכיבי הפיתוח בסעיף זה מוגדרים במרחב השמות הבא –
CW.Soloist.DataAccess.UnitOfWork

קובצי קוד המקור המכילים רכיבים אלו נגישים בקישור הבא –

<https://github.com/cwelt/Soloist/tree/master/DataAccess/UnitOfWork>

יחידת עבודה (Unit of Work) היא דפוס עיצוב ארכיטקטוני לניהול עדכוני זמן ריצה על אוסף יישויות עסקיות הרלוונטיות ל-Persistence. סנכרון שינויים אלו אל מול מדדים האחסן הגרפי, וכל זאת באמצעות מנשך High-level, בטרמינולוגיה של הישויות העסקיות בעולם האובייקטיבים, מבלי הצורך להתעסק או לדעת כיצד הישויות מאוחסנים בפועל (SQL/NoSQL/XML וכו').

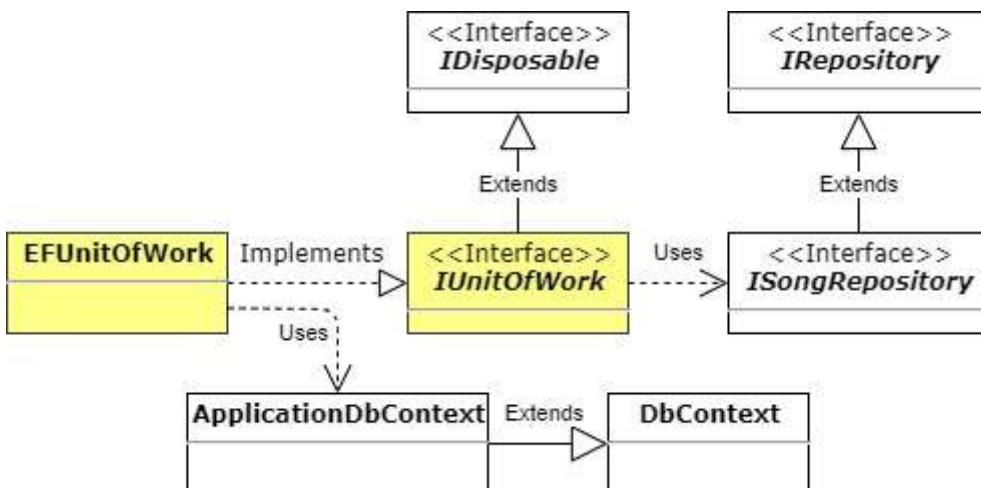
Repository אחראי על עדכון ניהול ישויות במאגר בזיכרונו זמן הריצה, וה-Unit of Work הוא זה שאחראי לשנכרון את תמונות המצב בין מאגר/ים שבזיכרונו זמן ריצה אל מול תמונות המצב במדדים האחסן הגרפי. כמו כן, אם ה-Repository מנהל בד"כ אוסף של ישויות במאגר מאותו טיפוס, ה-Unit of Work מנהל אוסף של אוסף ישויות, דהיינו אוסף של Repositories.

לשם כך Unit Of Work מגדר פועלות לרישום ומעקב סטטוס הישויות השונות, ופעולות לשימרת אוסף השינויים שהצטברו בזמן ריצה אל מול מדדים האחסן הגרפי. למשל פעולה רישום ישויות "מלולכות" – דהיינו רישום מופעים של אובייקטים בזמן ריצה חדשים ו/או מופעים שעודכנו מאז שנקראו ממדדים האחסן הגרפי, מודיעה ל-Unit of Work Unit-Commitment, או בכלל עדכון איזה קובץ XML.

מנקודת המבט של הקליינטים שעובדים עם Unit Of Work, הם רק צריכים לקרוא למתחוזות Commit כגון high-level, מבלי הצורך לדעת האם זה מבצע מבעלי הkulums פקודות SQL.

מיימוש דפוס ה-Unit Of Work במערכת בוצע תוך ניצול המימוש הקיים בחבילת ה-ORM של EntityFramework² : ישויות ה-DbSet של UnitOfWork מתפקדות בהתאם ל-Repositories, וישות ה-DbContext מתפקדת בהתאם ל-UnitOfWork. לפיכך, ניתן לחתוך קריאות ל-UnitOfWork ולהמאר אותו (להתאים אותו) לקריאות שה DbContext כבר מכיר ומבצע היבט, ובעצם להציג אליו את האחוריות. הערה : במערכת לא נעשה שימוש ישיר ב-DbContext אלא במחלקה ApplicationDbContext היורשת ממנו, ומרחיבה את הfonksiyonalitot שלה עם שירותים שמותאמים לישום Web-י וניהול רשומות משתמש.

להלן דיאגרמת מחלקה המתמקדת בקשר בין הישויות המרכזיות בתת-שכבה ה-Data Access – Unit Of Work Layer – מנקודת המבט של ה-Unit Of Work



² שימוש ב-EntityFramework Core גרסה 6, לא EntityFramework .NET Core
ניהול יחידות עבודה (Unit Of Work)

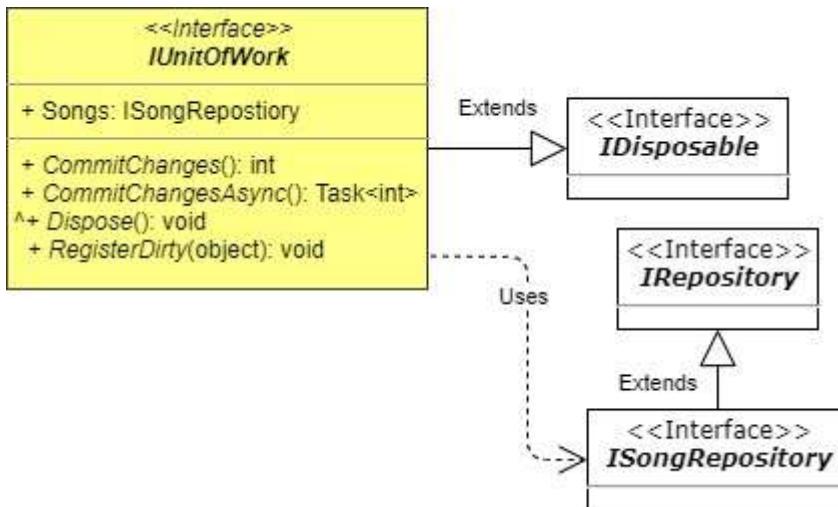
6.4.1 ממשק יחידת עבודה (IUnitOfWork Interface)

המנשך IUnitOfWork מגדיר ממשק עבור הפשטת פונקציונליות דפוס יחידת עבודה –

```
public interface IUnitOfWork : IDisposable { ... }
```

מנשך זה הוא למעשה נקודת הגישה של הקליניטים פונים לממשק ה-Database. קליניטים יכולים לקבל גישה למארגרים המנוהלים אצלם Properties, מעדכנים את המארגרים באמצעות המנשך שהם מספקים לאחזר ועדכו ישוות בזיכרון זמן ריצה, ולבסוף מבקשים מה-UnitOfWork לסנן את תמונה המצב של המארגרים שהוא מכיל אל מול תמונה המצב ב-Database.

להלן דיאגרמת מחלקה עבור המנשך IUnitOfWork המתארת את רכיביו המנשך –



מאפיינים: המנשך מכיל כמה פונקציות מוגדרי IRepository עבור כל הישויות העסקאות שהוא אחראי לסנן מול מסד הנתונים. המארגרים מסיעים לממשק ניהול המארגרים בזיכרון זמן ריצה של הישויות הקונקרטיות, כדי שהוא יוכל להתמקד במלאת הסנן מול מסד הנתונים. בשלב זה המאגר היחיד שמנוהל על ידי המנשך הוא מאגר שירים. אם בהמשך יוחלט לשמר במסד הנתונים ישוות נוספת – למשל שמירת המנגינות שנוצרות בתהליך הحلhana, לאחר יצירת טבלה מתאימה במסד הנתונים, מחלוקת מודול ייעודית ומימוש מאגר ייעודי עבור מנוגנות מוחנות, יש להוסיף את מנגנון המאגר של המנגינות לממשק UnitOfWork, כדי שייהיה זמין לקליניטים לאחזר ועדכו רשומות מנוגנות, וכן שייהיה תחת הפיקוח של UnitOfWork.

– IUnitOfWork: להלן תקציר המתוודות המוגדרות בממשק

Method	Description
CommitChanges	Updates the underlying database, either synchronously or asynchronously, and returns the number of successfully updated records in the underlying data store.
CommitChangesAsync	
RegisterDirty	Registers a tracked entity as dirty so its corresponding record would get updated in the database in the end of the unit of work when a request to commit the changes is made.
Dispose	Releases managed resources that are no longer needed.

קוד המקור של המנשך מוגדר בקובץ IUnitOfWork.cs שנitin נמצא בקישור הבא –

<https://github.com/cwelt/Soloist/blob/master/DataAccess/UnitOfWork/IUnitOfWork.cs>

6.4.2 מחלקה יחידת עבודה (EFUnitOfWork Class)

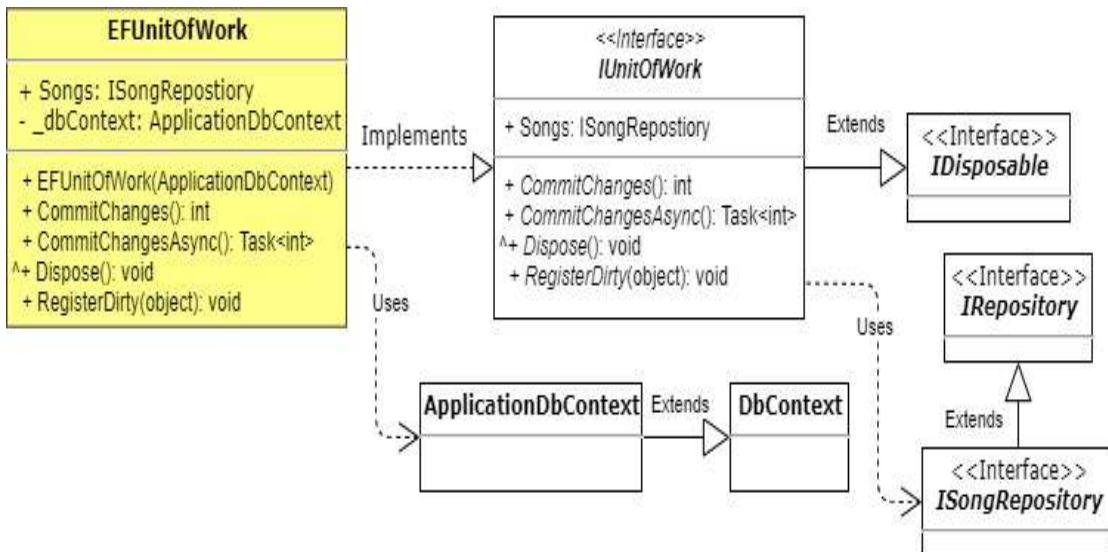
המחלקה EFUnitOfWork מimplements את הממשק IUnitOfWork של יחידת עבודה –

```
public class EFUnitOfWork : IUnitOfWork { ... }
```

ה-EF בשם המחלקה הם ראשי-תיבות של מבית Microsoft [ORM](#), [Entity Framework](#) חבילת ה- EF. שם המחלקה מרמז על האופן שבו היא מimplements את הממשק [IUnitOfWork](#): באמצעות הפונקציונליות שכבר מספקת "Out of the Box": החיבור מגדרה טיפוס המתפרק כמנהל ישות תוך שימוש הדפוס הארכיטקטוני [Unit of Work](#), ומחזק אוסף ישות גנריות מטיפוס [DbSet](#), אשר כל אחת מהן מתפרקת למעשה כ-Repository עבור הישות מהטיפוס הקונקרטי של אותו [DbSet](#). לפיכך, במקום כתוב מאפס את כל הלוגיקהلسנו כרונו ה-DB, ניתן בכך שימוש מותודות הממשק פשוט לבצע האצלת אחריות (delegation) אל ישות ה-EF, ובפרט אל ישות ה-[DbContext](#). ספציפית בפרויקט זה, לאחר שמדובר בישום אינטראקטיבי, נעשה שימוש בישות ה-[ApplicationDbContext](#) היורשת מ- [DbContext](#) ומרחיבה את הפונקציונליות בהתאם לישומי Web וניהול רשומות משתמש.

היתרונות בשימוש במנشك ובמחלקה מגשת זו עבור שימוש [UnitOfWork](#) במקומות שבו ישיר ב-EF בקוד בצד הקלינייטים (יתר שכבות המערכת) הוא האבסטרקציה: הקלינייטים אינם תלויים ב-EF אלא רק במנשיים המופשטים של ה-[IUnitOfWork](#) וה- [IRepository](#). התלות באבסטרקציה מאפשרת להחליף את השימוש הקיים של מחלקות אלו במחלקות אחרות, שמאפשרת את המשיים ע"י חיבור ORM אחרית ([NHibernate](#) למשל), או בכלל בשימוש ישיר של ספרייה ה-ADO.NET. כל עוד קיים שימוש למנשיים [IRepository](#) ו-[UnitOfWork](#), תוך שימוש [Factory](#)-[Pattern](#) מותאים (או מנגנון [Dependency Injection](#)), קלינייטים שעובדים מול המשיים בלבד (מול האבסטרקציה בלבד) המחלקות הקונקרטיות שמאפשרות את המשיים שkopoot לחנות.

להלן דיאגרמת מחלקה של EFUnitOfWork המתארת את רכיבי המחלקה והקשר שלה עם יתר המחלקות/מנשיים בתת-שכבת הגישה לנתונים –



המחלקה מגדרה ניחוח יחיד המקביל כפרמטר מופיע של יישות [ApplicationDbContext](#), שהוא כאמור יורשת מ-[DbContext](#) ומספקת בפועל את התמיכה בפונקציונליות הנדרשת לניהול ה-[UnitOfWork](#) באינטראקטיבי מול ה-DB. את המופיע הזה היא שומרת במשתנה הפרט [_dbContext](#).

המחלקה אינה מגדרה מאפיינים או מותודות נוספות מעבר לאלו שהוא מimplements מהמנشك [IUnitOfWork](#).

קוד המקור של המחלקה موجود בקובץ EFUnitOfWork.cs שנינו למצוא בקישור הבא – <https://github.com/cwelt/Soloist/blob/master/DataAccess/UnitOfWork/EFUnitOfWork.cs>

6.5 מיפוי בין אובייקטים לרשומות (EntityFramework ORM)

העבודה מול בסיסי הנתונים במערכת אינה מבוצעת ישירות עם ה-API של ADO.NET, אלא עם כלי ה-ORM – Entity Framework (Object-Relational-Mapping) של Microsoft- לישומי.NET..

כלי זה מממש טכניקת תכנות להמרה אוטומטית של נתונים מייצוגם כאובייקטים מורכבים בסביבת OOP לטבלאות שטוחות בסיס נתונים רלוונטי ולחפץ. לאחר הגדרת הקונפיגורציה של המיפוי, העבודה מול בסיס הנתונים היא מול המודל הקונספטוואלי בעולם האובייקטים, מוביל הצורך לדמת לעולם הפיזי של בסיס נתונים, מה שחווסף התעaskות Low-level כגון ניהול DB-Connection ל-DB, בKİיות בשמות טבלאות, עמודות, הרמת ייצוגים מתאפסים מערכות.NET CLR, SQL, ועוד. כל זאת מבוצע מאחרי הקלעים ע"י כלי -.ORM – בהקשר של מערכת זו, הכליל שנבחר הוא Entity Framework

התשתיית והקונפיגורציה של המיפוי בין בסיס הנתונים הפיסי למודל הקונספטוואלי בעולם האובייקטים (מחלקות ומנשכים) מרכזת מרחב השמות הבא (או בתתי-מרחבים שלו) –

`CW.Soloist.DataAccess.EntityFramework`

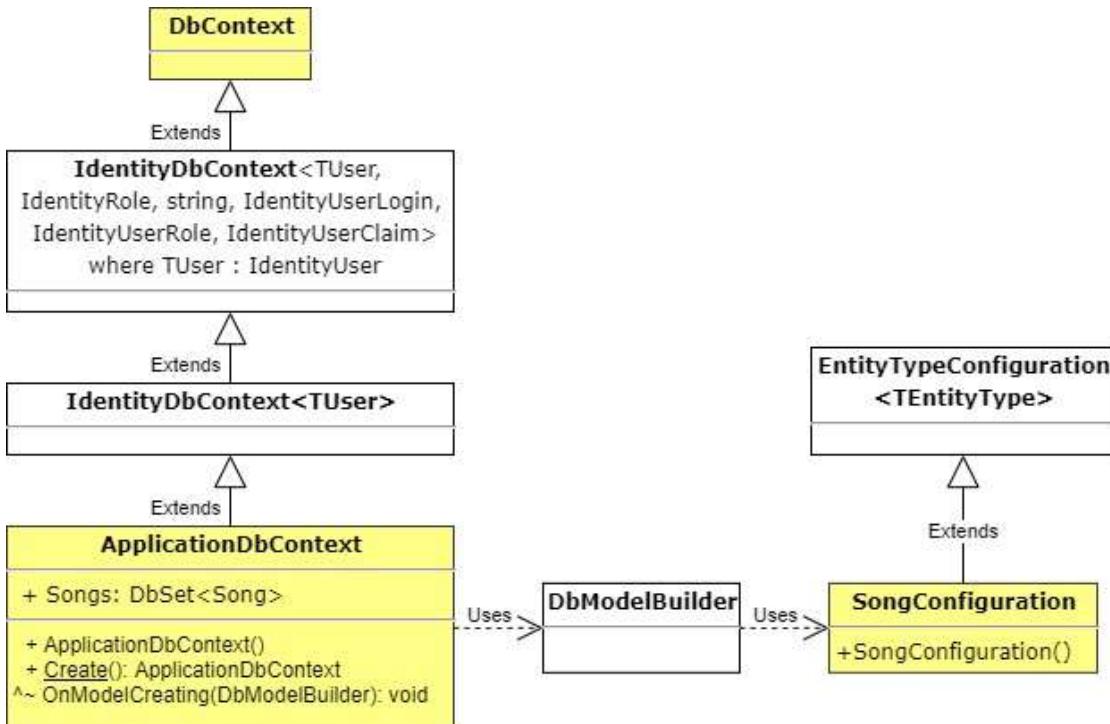
קבצי קוד המקור המכילים רכיבים אלו נגישים בקישור הבא –

<https://github.com/cwelt/Soloist/tree/master/DataAccess/EntityTypeConfiguration>

Entity Framework מאפשרת כמה גישות ודריכים להגדירה וניהול של קונפיגורציות המיפוי בין בסיס הנתונים הפיסי למודל הקונספטוואלי בעולם התכנות מונחה עצמים. הגישה שנבחרה במסגרת פיתוח המערכת היא נישת "Code First", שאננים דורשת יותר כתיבת קוד, ובפרט דרישת הגדרה ידנית מפורשת לא מחוללים אוטומטיים מבוססי XML, אולם הגדרה מפורשת זו בקוד שמבוצעת ידנית מספקת רמת שליטה טוביה יותר, ללא הסתמכות על תוכנות כלים אוטומטיים. הערה : זוהי גם הגישה היחידה הנתמכת ע"י Microsoft בגרסאות האחרונות של EF בפלטפורמת .NET Core : ב-.NET Core Entity Framework Core מובוצעת אך ורק ב-.Code First

הדרות המיפויים לא מבוצעות עם אנווטציות, אלא בקוד, באמצעות מחלקה `DbModelBuilder` שמקבלת מופע של מחלקה קונפיגורציות ייעודית, כגון `SongConfiguration`, המגדירה את קונפיגורציות המיפוי של ישות מחלקה המודול Song אל מול הטבלה התאימה לה בסיס הנתונים.

להלן דיאגרמת מחלקות המציגות (בצהוב) את המחלקות העיקריות במודול זה של EF ואת הקשרים ביניהם –



מיפוי בין אובייקטים לרשומות (EntityFramework ORM)

מחלקה ייחודית עבודה (EFUnitOfWork Class)

מחלקה קונטיקסט מסד הנתונים (ApplicationDbContext Class) 6.5.1

המחלקה ApplicationDbContext מנהלת את האינטראקציה מול מסד הנתונים הפיסי –

```
public class ApplicationDbContext : IdentityDbContext< ApplicationUser > { ... }
```

מחלקה זו יורשת ממחילה DbContext, המחלקה של Entity Framework שודאגת למשתמש את דפוס ה-Unit Of Work (סנכרון עדכוני זמן ריצה מול מסד הנתונים), ניהול ישויות שנמצאות במעקב ל-Persistence. תוך מתן גישה למאגרים שלהם בזמן ריצה בדמות מופע DbSet, ועוד. בהיררכיית עץ הירושה שבין DbContext ל-DbApplicationContext נמצא מספר מחלקות גנריות של ApplicationDbContext המוחיבות את הפונקציונליות הבסיסית עם פונקציונליות Authentication, Authorization & AAA. ייעודית לניהול משתמשים עם מימוש דפוס האבטחהASP.NET MVC ב-Web המפותחים ב-.NET Framework. הרחבות אלו חוסכות את הצורך בהגדירה מפורשת של טיפול ב-persistence של יישויות הקשורות לניהול משתמשים, כגון Chשבונות משתמשים, קבוצות הרשאה (Roles) ועוד.

מחלקה זו מגדרה אוסף של מאגרי ישויות – DbSet, שמספקים גישה לניהול אוסף ישויות מסוומות טיפוס במאגר זיכרונו זמן ריצה, ומתחזות לאתחול וקונפיגורציה של המיפויים –



להלן תקציר המתודות המוגדרות במחלקה, הראשונה – Create, היא מתודת סטטistica המשמשת את סביבת-ASP.NET MVC הסטנדרטיבית לאתחול וקונפיגורציה ראשונית של-EF, והשנייה – OnModelCreating, משמשת להגדרות מיפויים בקוד והוא נדרשת במחלקה זו לטובת הוספה מיפויים לישות של שיר ויישויות פרטניות למערכת נוספת במידת הצורך –

Method	Description
Create	Static factory method for creating ApplicationDbContext instances.
OnModelCreating	Maps table names, & relationships between the various entities.

המתודה OnModelCreating מקבל מופע של DbModelBuilder, שבתורו מקבל מופע של יישות קונפיגורציות מיפוי עבור איזשהו טיפוס קונקרטי. עבור כל טיפוס קונקרטי שכזה, יש להגדיר מחלקה נפרדת היורשת ממחילה הגנית EntityTypeConfiguration, להחילף בה את הטיפוס הגנרי בטיפוס הקונקרטי של המודול הרלוונטי, ולהגדיר בבניאי את כל המיפויים, כפי שמצוין לדוגמה במחלקה [SongConfiguration](#) (פירוט בהמשך).

קוד המקור של המחלקה מוגדר בקובץ ApplicationDbContext.cs שנitin נמצא בקישור הבא – <https://github.com/cwelt/Soloist/blob/master/DataAccess/EntityFramework/ApplicationDbContext.cs>

6.5.2 מחלקה מייפוי ישות שיר לטבלה (SongConfiguration Class)

– המחלקה SongConfiguration ממפה בין המחלקה Song לבין הטבלה המתאימה לה ב-DB.

```
internal class SongConfiguration : EntityTypeConfiguration<Song> {...}
```

מחלקה זו יורשת מהמחלקה הכלכלית `EntityTypeConfiguration<Song>`. המחלקה מדירה בוגר הבנאי שלה באמצעות יעוזות מיופיעים מול הטבלה המתאימה ב-DB – שם הטבלה, מייפוי מאפיינים (Properties) מול עמודות בטבלה, הגדרה איזה מאפיינים במחלקה אינם רלוונטיים למייפוי ושמירה ב-DB (כגון מאפייני ניוט), הגדרת אילוצים על העמודות (כגון הגבלת אורך מחרוזת בעמודות varchar), סימון המאפיינים שמרכיבים את המזזה החד-ערך (מפתח) בטבלה, הגדרה האם מותרים ערכי null, וכן הגדרות נספנות רלוונטיות למיפוי בין הישות הקונספואלית המוצגת כמחלקה לבין הישות הפיסית ב-DB המוצגת כרשומה שטוחה בטבלה בסיס נתונים.

המודעות היודוות להגדרת הקונפיגורציה מחזירות כולל איזשהו אובייקט קונפיגורציה, כך שנייתן לבצע הגדרה ב-`"Fluent Api"` ולשרשר קרייאות קונפיגורציה זו אחר זו בדומה לדפוס העיצוב Decorator, שכן כל מודול קונפיגורציה מחזירה אובייקט קונפיגורציה מעודכן שאותו אפשר להמשיך ולקנג.

– קוד המקור של המחלקה מוגדר בקובץ `SongConfiguration.cs` שנitin נמצא בקישור הבא
<https://github.com/cwelt/Soloist/blob/master/DataAccess/EntityFramework/MappingConfigurations/SongConfiguration.cs>

6.6 מיגרציות (Migrations)

בשיטת ההגדרה של Entity Framework העובדת בהגדורות מפורשות בקוד (code first), שינויים על הקוד המשפיעים על מסד הנתונים מייצרים מיגרציות (בהנחה שמיגרציות אפשרו בפרויקט עם הרצת הפקודה `Enable-Migrations` מתוך ה-Console). מיגרציות אלו הן מחלקות היורשות מ-`DbMigration`, והן מכילות שתי מוגדרות: הריאונה – `Up`, האחראית לשדרוג או להעלות גרסה במסד הנתונים, דהיינו להביא אותו למצב העדכני כפי שמוצג במודול הקונספואלי במחלקות בעולם התכונות מונחה עצמים, ואילו המודעה השניה – `Down`, האחראית על ביצוע רגרסיה – מעין `Rollback` לגרסה קודמת, לפיכך מוגדרות אלוuireות להציג פועלות הופכיות זו לזו, למשל אם `Up` מכילה הוראה להקים טבלה, אז `Down` צריכה להכיל הוראה למחוק טבלה זו (`Drop`). יש לחתות בחשבונו שהגדרת פועלות הופכיות יכולה לגרום לאיבוד נתונים אם לא נזהרים: למשל אם המצב הוא הפרך – ה-`Up` גורם למחיקת טבלה וה-`Down` גורם ליצירת הטבלה מחדש, יש לדאוג טרם מיחיקת הטבלה לגבות את הנתונים (תוכן הטבלה) ולטעון אותם מחדש במסגרת פועלות ה-`Down`.

מחלקות המיגרציה השונות נוצרות אוטומטית (שוב, בהנחה שהן מאפשרות בפרויקט): מנגנון ה-`EF` בודק האם השינויים שבוצעו במחלקות משפיעות על מסד הנתונים באמצעות הגדרות הקונפיגורציות וטבלת היסטורית דלתאות ייעודית במסד הנתונים, ובהתאם לכך מחולל אוטומטית פקודות רלוונטיות בהתאם לצורך במחלקות המיגרציה שבתורן מתורגמות אח"כ לסקרייפט SQL שמורץ מול ה-`Database`. חלק מהמיגרציות הותאמו ועובדנו ידנית. ככל מרוכזות – במרחב השמות הבא

`CW.Soloist.DataAccess.Migrations`

בפועל הן שייכות כלן ל-`Entity Framework`, ולכן יותר אולי היה לשמור אותם במרחב השמות של `EntityFramework`, אולם כברירת מחדל הן נשמרות אוטומטית בספרייה ייודית זהה מרחב השמות נתון להם בהתאם. קוד המקור שלהם זמין בקישור הבא –

<https://github.com/cwelt/Soloist/tree/master/DataAccess/Migrations>

7. שכבת התצוגה (PL)

שכבה התצוגה (PL) – Presentation Layer – אחראית על האינטראקציה מול משתמשי הקצה – הגדרת משקל ממשק גרافي (GUI), האזנה לקלט המשמש ומtran מענה לבקשת שלו תוך אינטראקציה זו עם שכבת הגישה לניטונים לאחזר ועדכו רשות ב-DB, והן שכבת הלוגיקה העסקית של האפליקציה לטובת צרכית שירוטי הלחנה שהמשתמש מעוניין בהן.

שכבה זו מוגדרת בשלמותה כספרייה בפרויקט Web ייעודי המהווה יחידת assembly עצמאית: WebApplication. בהתאם (ועיף [מוסכמת השמות במערכת מרוחבי שמות](#)), כולל מרוחבי השמות CW.Soloist.WebApplication. השיעיכים לשכבה זו מושרים תחת מרחב השמות (namespaces)

פרויקט ה-Web כתוב בפלטפורמת MVC³.ASP.NET. פלטפורמה זו מבוססת על בסיס דפוס העיצוב הארכיטקטוני שלל שמה היא נקראת **MVC**: Model-View-Controller – ה-Model: מודול נתונים POCO המציגות את היחסיות העסקית, ה-Views הם החזית שהמשתמש רואה – הם מחלקות HTML הנוצרים לאחר עיבוד קבצי cshtml המורכבים מתערובת של C#, HTML, Javascript, Razor ותחביר ייעודי של מנוע HTML.NET MVC של ASP.NET MVC שמנתח קבצי Controllers, מייצר מהם קבצי HTML לאחר שיובץ נתונים רלוונטיים מהמודלים, ולבסוף ה-HTML, אליהם בקרים המטפלים בבקשת משתמשי הקצה, מעבדים את הקלט, אוסףים נתונים רלוונטיים ומרכזים אותם במחלקות המודלים אותם הם מעבירים ל-Views לתצוגה ולבסוף מחזירים אותו כפלט למשתמש, שמקבל דף HTML עם התוכן הנדרש.

הפלטפורמה כבר מספקת מסגרת שלמה ליישום ה-Web-י, שבה ניתן להגדיר מודלים, דפי תצוגה ובקרים, וכן ונקודות עוגן נוספת שבחן ניתן להתערב ולדרוס או להרחב את לוגיקת ברירת המחדל הקיימת בפלטפורמה. מסמך זה מתמקד רק ברכיבי הפיתוח ונקודות ההתערבות שבוצעו לטובת המערכת, ולא מפרט את הקוד המקורי של Microsoft שכבר מגע מוקן במסגרת הפלטפורמה. תיעוד של קוד זה ניתן למצוא באתר של Microsoft

<https://docs.microsoft.com/en-us/aspnet/mvc/overview/getting-started/introduction/getting-started>

להלן תקציר מבנה עצם התיקיות של הפרויקט –

תיקיה	תוכן
App_Data	נתונים נשמרים במערכת הקבצים של שרת ה-Web, דוגמת לוגים, מסמכים PDF, שזמינים להורדה מאתר האפליקציה, וקבצי MIDI והטקסט של השירים.
App_Start	מחלקות עם שגרות קומפונטיזציה המופעלות עם אתחול האפליקציה, כגון רישום שירותים ומטודות/מחלקות שימושו את האפליקציה בנקודות שונות.
Content	משאים סטטיים כמו קבצי CSS מספרי Bootstrap עבור עיצוב התצוגה.
Controllers	בקרים – המחלקות שמתפלות בבקשת השינויים המגיעות מהמשתמש.
Filters	מחלקות המגדירות פונקציונליות פרטנית שמרחיבה/דורשת את הפונקציונאליות הסטנדרטיבית של NET. בנקודות עוגן מוגדרות מראש להתרבות, כגון הגדרת הלוגיקה לטיפול ב-Exception שלא נתפס ע"י סביבת זמן הריצה.
fonts	sett פונטיים מוקן שמשמעותם אוטומטית עם הקמת פרויקט ASP.NET MVC.
Scripts	sett סкриיפטים מוקן ב-JQuery ו-Bootstrap מספריites שמגייעו אוטומטית עם הקמת פרויקט ASP.NET MVC.
Validations	מחלקות ולידציה שנitin להפעיל על שדות בטפסי HTML באמצעות אנטציזות.
ViewModels	מחלקות DTO (Data Transfer Objects) ייעודיות ל-Views. אלו הם למעשה פשט מודלים שנוצרו בהתאם אישית ל-Views.
Views	דפי Razor (cshtml) המתורגמים לדפי HTML מושערים בתונינים.
/ (root)	משאים וקבצי קומפונטיזציה גלובליים כגון packages.config – קובץ ה-XML לניהול תלויות מספריות חיצונית (בדומה לקובץ POM ב依üşומי Java שעוברים הידור עם Maven, Global.asax, Global.asax.cs) לרשום גלובלי של מחלקות לטיפול באירועים אפליקטיבים, favicon.ico – האיקון המוצג בדף עברו דף האפליקציה.

הסיעיפים הבאים מפרטים את כל הפיתוחים והרחבות שבוצעו בפרויקט ה-ASP.NET MVC.

7.1 **תצורת התוכן האפליקטיבי על השרת (App_Data)**

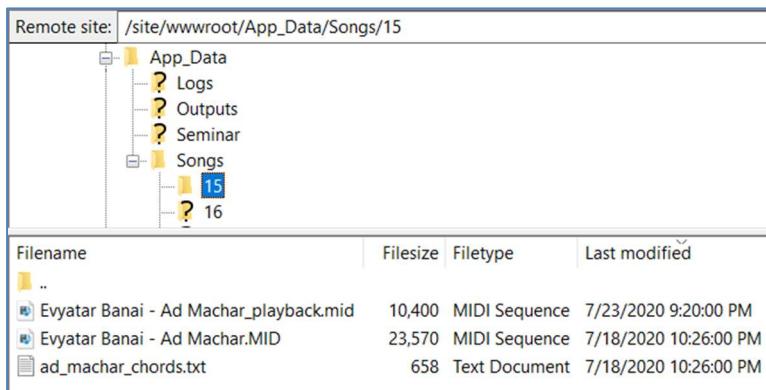
התיקייה App_Data השמורה על השרת מכילה מסמכים וקבצים שימושים את האפליקציה. תייקייה זו מכילה 4 תת-תיקיות : Outputs,Songs,Logs ו-Seminar.

לוגים (Logs) 7.1.1

התיקייה Logs מכילה קבועים שהמערכת רושמת לתוכם לצורכי בקרה: למשל –
מכלול פירוט שגיאות Exception בלתי צפויים שהתקבלו, ו- RequestLog.txt מתעד
פרטי בקשות של משתמשים באתר.

שירים (Songs) 7.1.2

התיקייה Songs מכילה את קבצי השירים שפרטיהם מאוחסנים ב-DB. הפרטים של השירים מונחים ב-DB, אולם הקבצים עצם שמורים על השרת – אלו הם קבצי MIDI וקבצי טקסט שכםוט ואינם צורכים זיכרון. הקבצים של כל שיר נשמרים בתת-תיקייה מסוימת בתיקיה Songs. כאשר שיר חדש נשמר ב-DB, הוא מקבל מספר מזהה חד-ערכי: Id. המערכת יוצרת עבורה תיקייה חדשה בתיקייה Songs תחת Id זה, ומרכזת שם את הקבצים הרלוונטיים לשיר. מצ"ב לדוגמה תצלום מהתוכנה FileZilla המשמשת כ-FTP Client לחיבור מרוחק לשם העברת קבצים בין תחנת העבודה לשרת ה-Web. בתצלום המוצג ניתן לראות בחלק העליון את מבנה תיקיית Songs על השרת כפיה שתואר לעיל, ובחלק התיכון את קבצי השיר שה-ID שלו הוא 15 –



–DB, ברשומות השיר רשומים שמות הקבצים השונים (קובץ אקורדים, קובץ MIDI וקובץ פלייבק) שבתוכה הקיימת של השיר, כך שהמערכת יכולה להרכיב את הנתיב המלא לקובץ ע"י שרשור הנתיב לתיקיית השיר עם שם הקובץ הרלוונטי. להלן שמות הקבצים ב-DB –

```
SQLQuery1.sql - den...b (soloistdb (213))* ▾ ×  
SELECT Id, MidiFileName, ChordsFileName, MidiPlaybackFileName  
FROM dbo.Songs WHERE Id='15';  
131 % ▾  
Results Messages  


|   | Id | MidiFileName                  | ChordsFileName       | MidiPlaybackFileName                   |
|---|----|-------------------------------|----------------------|----------------------------------------|
| 1 | 15 | Evyatar Banai - Ad Machar.MID | ad_machar_chords.txt | Evyatar Banai - Ad Machar_playback.mid |


```

מנגינות (Outputs

תיקייה Outputs מכילה את קבצי ה-MIDI עם מגוון הפלט שהמערכת מלחינה. המערכת שומרת שם את הפלטים לפני שהיא מורידה אותם למחשב המשמש. בדומה לתיקייה Songs, גם תיקייה זו מרכבת את הקבצים השונים לפי מספר ה-PI של השיר.

7.1.4 סמינר (Seminar)

תקיימת Seminar מכילה מסמך עבורות סמינר בנושא הלחנה עם אלגוריתמים גנטיים. עבורה זו מכילה את הבסיס התאורטי לIMPLEMENTATION המרכז במערכת. המסמך זמין להורדה באתר האפליקציה וכן שמור ישירות על השרת לטובת גישה מהירה.

7.2 קונפיגורציות אתחול אפליקציה (App_Start)

בתיקייה זו מרכזות מחלקות שונות המכילות שגרות ושירותים לביצוע/רישום עם עליית האפליקציה. סעיף זה מתרח את העדכונים שבוצעו בתיקייה זו עבור המערכת.

7.2.1 הגדרת חבילות (BundleConfig)

המחלקה BundleConfig מכילה הגדרות לאירוע של קובצי javascript ו-css לחבילות. לטובת עדכון עיצוב בירית המחדל של bootstrap המוגדרת ב-ASP.NET MVC, קובץ ה-bootstrap.css מעתוסף בברירת מחדל ל-template הוחלף ב-bootstrap-darkly.css. קובץ זה הורד לאתר bootstrap ונשמר בתיקיית המשאבים הסטיטיים Content בתת-תיקייה Themes.

7.2.2 רישום מחלקות פילטר (FilterConfig)

המחלקה FilterConfig מכילה רישום הגדרות מחלקות המגדירות לוגיקה פרטנית מותאמת אישית שמחילה/מרחיבה את לוגיקת בירית המחדל הסטנדרטיב שמסופקת ע"י פלטפורמת ASP.NET MVC. במחלקה זו מבוצע רק רישום מחלקות אלו, ובגוף המחלקות עצמן נרשומות נקודות העוגן שבון הן מעוניינות להתעורר – למשל התערבות לפני יצוא מתודה ע"י בקר. מחלקות אלו נקראות מחלקות פילטר, שכן בהתערבויות התעבורת עוברת דרכם כמו פילטר, והן יכולות לח吉利 מה לעשות עם התעבורת והאם לסתן לחסום/לאפשר וכך' בדומה ל-Firewall. ניתן למשתתף בירית מחלקה פילטר שבודקת האם התעבורת מגיעה ממנווע חיפוש (Web crawler engine) ובמידה וכן היא תסנן את הבקשה ותחזיר שגיאה כגון 404 (Not Found). להלן מחלקות הפילטר שהמערכת מדירה בעצמה ורשות מחלקות פילטר במסגרת FilterConfig.cs.

– Filters

Filter Class	Purpose
CrawlerFilter	Block access to requests that come from web browser crawlers
LogRequestFilter	Log all the HTTP requests that are send to the web server.

7.2.3 הגדרת שירותי זהיות (IdentityConfig)

הקובץ IdentityConfig.cs מרכז מחלקות שירותות מסוימות בתהליכי זהיות וアイמות של משתמשים, כגון מחלקות לקביעת מדיניות אבטחה לגבי הסיסמה ושירותי שליחת מייל ו-sms למשתמש ששכח סיסמה ו/או אמציע זיהוי נוספים עבור two-factor authentication. מחלקות אלו מגייעות אוטומטית עם ה-ASP.NET MVC אם בוחרים באפשרות של ניהול זהיות בעת הקמת הפרויקט, עם זאת, המחלקות לא מכילות פונקציונליות שלמה. בפרט – עבור שירות שליחת מייל ו-sms, פלטפורמת ASP.NET MVC מספקת מחלקות שירות עם מתודות ריקות המתפקידות כ-"placeholders", נקודות עוגן שאוון יש למשתמש עצמאית.

המערכת ממשת בשלב זה את שירות שליחת המייל, שירות זה מוגדר במחלקה EmailService במסגרת המתודה SendAsync. הלוגיקה של שליחת המייל מוגדרת במתודה נפרדת שנראית מתוך מתודת השירות SendAsync. כרגע מוגדרים שני שימושים שונים, האחד ממומש באמצעות שירותי SMTP הבסיסיים המספקים ע"י .NET : SendEmailWithSMTP, SendEmailAsyncWithSendGridService : SendGridService : SendEmailAsyncWithSendGridService. נכון מומומש לערךיו המתודה SendAsync מוגדרת להפעיל את מתודת שליחת המייל בימוש של SendGrid.

7.2.4 הגדרת שירותי הרשאה (Startup.Auth)

המחלקה Startup מגדירה שירותי הרשאה שונים, ביניהם האפשרות לבצע אימות חשבון מול אפליקציות צד ג' באמצעות פרוטוקול [OAuth 2.0](#). בשלב זה מוגדרת האפשרות לבצע אימות שכזו מול חברות Google, ע"י קרייה למתודת UseGoogleAuthentication על מופע ה- IAppBuilder תוך אספקה פרטיה הזדהות של האפליקציה שהתקבלו ברישום מול Google. פרטים אלו מוחסנים בקובץ קונפיגורציית ייעודי על השרת שאינו חשוף באופן ציבורי – AppSensitiveSettings.config.

7.3 בקרים (Controllers)

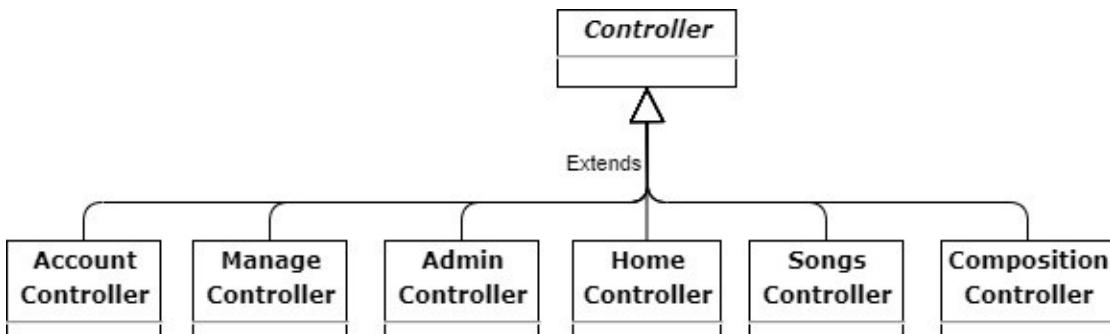
בתיקיית Controllers מרכזות כל מחלקות הבקרים. בהתאם, הן מוגדרות ב-namespace `CW.Soloist.WebApplication.Controllers`

קוד המקור של כל מחלקות אלו זמין בקישור הבא –

<https://github.com/cwelt/Soloist/tree/master/WebApplication/Controllers>

סעיף זה מפרט את מחלקות הבקרים השונות. בהתאם למבנה של MVC, כל בקר מוגדר במחלקה נפרדת משלו היורשת ממחלקה אב האבסטרקטית `Controller`, ושם המחלקה שלו היא שם הבקר בתוספת הסיומת `.Controller`. למשל `SongsController` הוא שם מחלקת בקר השיריים.

להלן דיאגרמת מחלקה בסיסית המתארת את הבקרים השונים ואת קשר הירושה שלהם למחלקה-האב של הבקר האבסטרקטי –



להלן תקציר הבקרים השונים –

Controller	Description
Account	ASP.NET MVC predefined controllers for managing the AAA security pattern: Authentication, Authorization & Accounting.
Manage	
Admin	Responsible for handling requests for application administrator privileged user for managing the application.
Home	Responsible for handling requests for and from the application home page, and supplying basic general internal services to other controllers.
Songs	Responsible for managing song resources: handling requests for uploading new songs, updating songs, deleting songs and querying songs.
Composition	Responsible for composition requests, validating input, gathering resources, integrating with relevant services and returning result to end users.

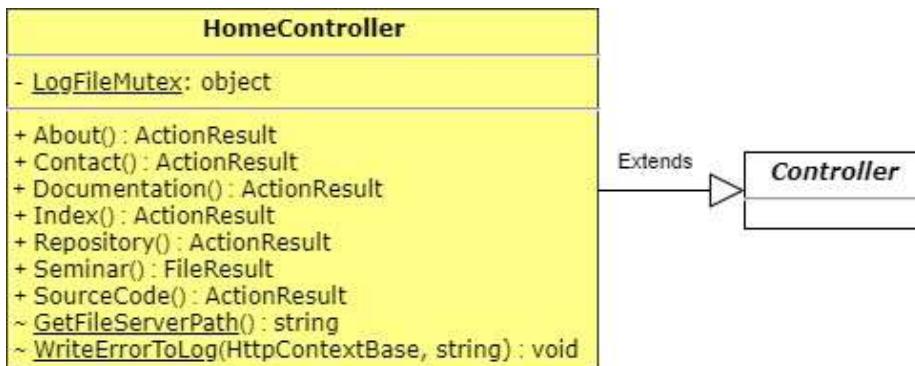
7.3.1 בקר דף הבית (HomeController)

מחלקה `HomeController` מטפלת בבקשתות המגיעות מותוך הבית של האתר האפליקציה וכן בבקשתות שירות כליליות שRELONETWORK מצלל הבקרים כגון קבלת נתיב הספרייה –

```
public class HomeController : Controller {...}
```

7.3.1.1 דיאגרמת מחלקה

להלן דיאגרמת מחלקה של בקר דף הבית HomeController



7.3.1.2 מאפיינים

המחלקה מגדרה שדה יחיד – מנעול למניעת הדדיות של גישה לקובצי לוגים במקביל –

```
private static readonly object LogFileMutex = new object();
```

7.3.1.1 מתודות

להלן חתימות המתודות הציבוריות המוחזירות עם תקציר תיעוד בסיסי שלhn –

```

public ActionResult Index() {...} // Returns Home page
public ActionResult About() {...} // Returns About page
public ActionResult Documentation() {...} // Returns Documentation page
public ActionResult SourceCode() {...} // Returns Source Code page
public ActionResult Repository() {...} // Redirects to source code repository
public ActionResult Contact() {...} // Returns Contact Details page
public FileResult Seminar() {...} // Downloads the seminar paper document
public ActionResult PrivacyPolicy() {...} // Displays privacy policy statement
  
```

על פ' קונבנציית ברירת המחדל, מתודות אלו נקראות ע"י נתב האפליקציה בעת גישה לנטייב של דף הבית (שם הבקר) עם שם המתודה (שם ה-Action). למשל: ה-URL הבא מנותט לבקר דף הבית וקורא למתודת Documentation שלו: <ApplictionURL>/Home/Documentation. על השרת שכרגע מארח את האפליקציה, הנתיב הוא –

<https://soloist.gear.host/Home/Documentation>

בנוסף למתודות הציבוריות המוחזירות דפים כתגובה לגילישה של משתמש ל-URL מתאים, בקר דף הבית מגדר שם שני מתודות שירות כליליות לשימוש פנימי בין הבקרים: הראונה מוחזירה את הנתיב לתקייה על השרת שבה שומרם הקבצים (App_Data), והשנייה משמשת את הבקרים לרשום שגיאות בקובץ לוג השגיאות –

```

internal static string GetFileServerPath(HttpContextUtilityBase server = null){...}
internal static void WriteErrorToLog(HttpContextBase context, string errorMessage) {...}
  
```

7.3.2 בקר שירים (SongsController)

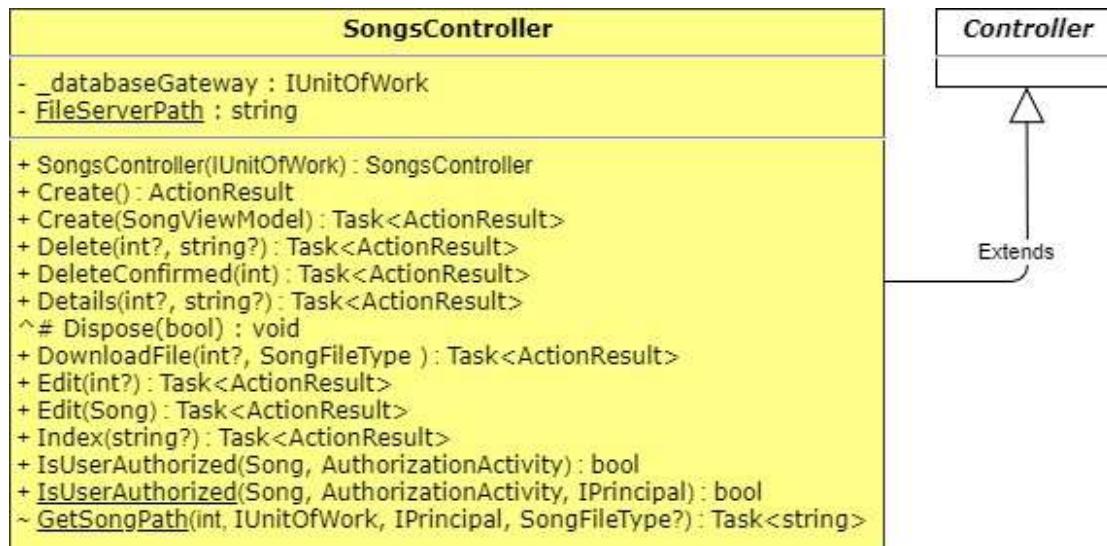
המחלקה SongsController אחראית לטפל בבקשת ניהול מאגר השירים באפליקציה –

```
public class SongsController : Controller {...}
```

במסגרת האחריות שלה המחלקה מספקת תמיכה בפעולות CRUD (יצירה, אחזור, עדכון ומחיקה) של משאבי שירים במערכת, בכפוף לבדיקה הרשות המשמש. היא אחראית בין היתר – על מסך אינדקס השירים. להלן קישור למסך זה על גבי השרת הנוכחי המארח את האפליקציה – <https://soloist.gear.host/Songs>

7.3.2.1 דיאגרמת מחלקה

להלן דיאגרמת מחלקה של בקר השירים – SongsController



7.3.2.2 מאפיינים

המחלקה מגדרה שני מאפיינים – ידית גישה לאינטראקציה מול ה-DB, ונתיב לתקינה על השרת שבה מנהל מאגר קבצי השירים –

```

private IUnitOfWork _databaseGateway;
private readonly static string FileServerPath = HomeController.GetFileServerPath();
  
```

7.3.2.3 בניאי

המחלקה מגדרה בניאי המציג את התלות שלו במופע מנהל ישוות DB ביחידת עבודה –

```

public SongsController(IUnitOfWork unitOfWork) { _databaseGateway = unitOfWork; }
  
```

מופע של `IUnitOfWork` "מושתל" פנימה אוטומטית ע"י מנגןו ה-DI (Dependency Injection) של AutoFac, ע"פ הגדרת התלות בקומפיגורציה ה-IoC (Inversion Of Control). בקובץ `Global.asax` בואפן זה, ניתן להחליף את מימוש ה-DB לחלווטין באופן שקוף לבקר בישום ה-.Web, כל עוד המימוש החדש מקיים את דרישות הממשק `IUnitOfWork`.

מתודות 7.3.2.4

להלן חתימות המתודות במחלקה –

```

public ActionResult Create() {...}
public async Task<ActionResult> Create(SongViewModel songViewModel) {...}
public async Task<ActionResult> Delete(int? id, string errorMessage = null) {...}
public async Task<ActionResult> DeleteConfirmed(int id) {...}
public async Task<ActionResult> Details(int? id, string message = null) {...}
protected override void Dispose(bool disposing) {...}
public async Task<ActionResult> DownloadFile(int? id, SongFileType songFileType) {...}
public async Task<ActionResult> Edit(int? id) {...}
public async Task<ActionResult> Edit(Song updatedSong) {...}
internal static async Task<string> GetSongPath(int songId, IUnitOfWork db, IPrincipal
    user, SongFileType? songFileType = null) {...}
internal async Task<string> GetSongPath(int songId, SongFileType? songFileType = null) {...}
public async Task<ActionResult> Index(string message = null) {...}
public static bool IsUserAuthorized(Song song, AuthorizationActivity authActivity,
    IPrincipal user) {...}
public bool IsUserAuthorized(Song song, AuthorizationActivity authActivity) {...}

```

להלן תקציר תיעוד בסיסי של המתודות שלעיל –

Method	Description
Create	Handles HTTP GET & POST requests for creating a new song: GET retrieves the html form and sends it to the user, and POST sends the filled html form back to the server.
Delete	Handles HTTP GET & POST requests for deleting an existing song: GET retrieves the html form and sends it to the user, and POST sends the filled html form back to the server.
DeleteConfirmed	
Details	Handles a HTTP GET request for displaying a detailed screen for a specific song.
Dispose	Releases managed resources that are no longer required.
DownloadFile	Downloads a requested song file to the end user's computer.
Edit	Handles HTTP GET & POST requests for editing an existing song: GET retrieves the html form and sends it to the user, and POST sends the filled html form back to the server.
GetSongPath	Gets the path of the given song's resources on the hosting file server.
Index	Gets and displays song list that user is authorized to see.
IsUserAuthorized	Checks if a user is authorized for the given song and activity.

7.3.3 בקר הלחנה (CompositionController)

– המחלקה CompositionController אחראית לטפל בבקשת להלחנת מנגינות חדשות –

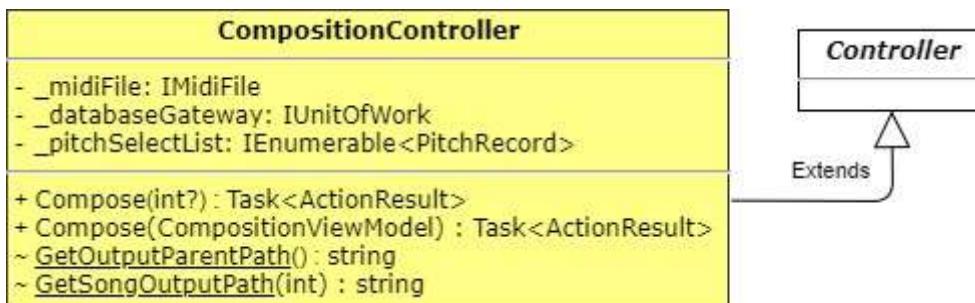
```
public class CompositionController : Controller {...}
```

להלן קישור (על השרת הנוכחי המארח את האפליקציה) למסך טופס הלחנה שהבקר מנהל –

<https://soloist.gear.host/Composition/Compose>

7.3.3.1 דיאגרמת מחלקה

להלן דיאגרמת מחלקה של בקר הלחנה של CompositionController



7.3.3.2 שדות

המחלקה מגדרה שלושה שדות –קובץ MIDI של פلت המנגינה שתיווצר בתהליך הלחנה, ידית

גישה לאינטראקציה מול ה-DB, ורכף פנימי של צלילים לטובת מנשך המשמש –

```
private IMidiFile _midiFile; // used for the composed melody output file
private IUnitOfWork _databaseGateway; //abstract gateway to the underlying database
private IEnumerable<PitchRecord> _pitchSelectList; // structures pitch data to view
```

7.3.3.3 בניאי

המחלקה מגדרה בניאי המציג את התלות שלו במופע מנהל ישות DB ביחידת עבודה –

```
public CompositionController(IUnitOfWork unitOfWork) { ... }
```

מופע של IUnitofWork "מושתל" פנימה אוטומטית ע"י מנגנון ה-DI (Dependency Injection) של AutoFac, ע"פ הגדרת התלוויות בקונפיגורציית IoC (Inversion Of Control) (בקובץ Global.asax). באופן זה, ניתן להחליף את מימוש ה-DB לchlוטין באופן שקווי לבקר ביישום ה-.Web, כל עוד המימוש החדש מקיים את דרישות המنشך IUnitofWork.

7.3.3.4 מתודות

להלן חתימות המתודות במחקה –

```
public async Task<ActionResult> Compose(int? songId = null) {...} // HTTP GET
public async Task<ActionResult> Compose(CompositionViewModel model) {...} //HTTP POST
internal static string GetOutputParentPath() {...} // output root folder
internal static string GetSongOutputPath(int songId) {...} // song output folder
```

להלן תקציר תייעוד בסיסי של המתוודות שלעיל –

Method	Description
Compose	Handles HTTP GET & POST requests for composing new melodies GET retrieves the html form and sends it to the user, and POST sends the filled html form back to the server.
GetOutputParentPath	Internal helper methods for retrieving the paths on the server to the midi output folders: parent is the global root output path, and song output path is the local output path for a specific song.
GetSongOutputPath	

7.3.3.5 מחלקה רשומה גבוהה צליל (PitchRecord Class)

המחלקה PitchRecord היא מחלקה פרטית פנימית המكونת במחלקה בקר השירים, שמשמשת להגדרת מבנה רשומה עוזר מותאמת למשק המשתמש –

```
private class PitchRecord {
    public NotePitch Pitch { get; set; }
    public string Description { get; set; } }
```

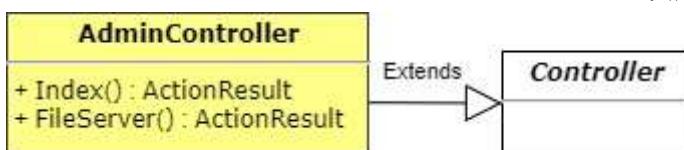
7.3.4 בקר ניהול (AdminController)

המחלקה AdminController אחראית לטפל בבקשתות מנהלו ניהול האפליקציה –

```
public class AdminController : Controller {...}
```

הבקר חושף את משק ניהול הנהלה רק עבור משתמשים מורשים. משק זה זמין בקישור הבא (על השרת הנוכחי המארח את האפליקציה) :

להלן דיאגרמת מחלקה עבור בקר ניהול –



בשלב זה משק ניהול מספק רק שירותים בסיסי של הפניה לשרת הקבצים של השירות. שירות*י ניהול נוספים כגון ניהול רשומות משתמשים מבוצעים בשלב זה ע"י עדכון ישיר של הרשומות הדרולונטיות מתוך מערכות DBMS. עם זאת, הקמת משק ניהול מהוועה תשתיית להרחבות עתידיות. אלו מתוודות המחלקה –

```
public ActionResult Index() {...} // Displays administration page
public ActionResult FileServer() {...} // Redirects to file server
```

7.3.5 בקרי ניהול חשבונות משתמשים

המחלקות ManageController ו- AccountController המנהלות את חשבונות המשתמשים מסופקוות ע"י הפלטפורמה הסטנדרטית של ASP.NET MVC. וモトイידות באתר Microsoft. העדכון היחידי שבוצע מחלקות אלו הוא הפניה לקונטיקט Entity Framework משכבות הגישה לנוטונים במקום זה שהגיע כברירת מחדל בפרויקט ה-Web שהוסר למניעת כפיפות קונטיקטים.

7.4 פילטרים (Filters)

פילטרים הם מחלקות המגדירות לוגיקה לביצוע בנקודות עוגן מוגדרות מראש ש-ASP.NET MVC מימושה. הרעיון דומה לדפוס העיצוב Template Method הפלטפורמה מגדרה את המנגנון וספקת מימוש ברירת מחדל בנקודות עוגן (לעתים זהו מימוש ריק), ומאפשרת לנו המפתחים לזרום את ברירת המחדל ולספק מימושים שלנו בנקודות החתurbות.

פלטפורמת ASP.NET MVC מספקת כבר ארבעה סוגי פילטרים OOB (Out Of the Box), העונים לצרכים שונים וספקים נקודות התערבות שונות. להלן ארבעת סוגי פילטרים אלו והמנשך שעלייהם ממש –

#	Filter Type	Interface This Filter Must Implement
1	Authorization Filter	IAuthorizationFilter
2	Action Filter	IActionFilter
3	Result Filter	IResultFilter
4	Exception Filter	IExceptionFilter

הפילטר הראשון (Authorization) מטפל בסוגיות אבטחה ויכול לחסום ניסיונות גישה לא מורשים (למשל אם משתמש מתחכם מנסה לגשת ל-URL שאמור להוביל לפועלם בקר שמצריכה הזדהות מוקדמת). ה필טר השני (Action) עוטף בקשות שגיאות אל הבקר, וספק נקודות התערבות לפני ביצוע הפעולה ע"י הבקר ולאחריה. ה필טר השלישי (Result) מספק נקודות התערבות לעיבוד הערך המוחזר מפעולת הבקר, לפני ואו אחרי העברתו בחזרה למשתמש, ואחרון חביב (Exception), מספק נקודות התערבות לטיפול בחירוגות שלא נתפסו ע"י האפליקציה.

כל הפילטרים יירושים ממחלקת האב `AuthorizationAttribute`. כאמור, ניתן להפעיל אותם באופן מקומי על בקרים/מתודות ע"י סימונים באנווטציה מתאימה. למשל, פעולה הבקר של העלאת שיר חדש למערכת סומנה באנווטציה `Authorize` המנacha את פילטר ה-`Authorization` לחסום פניות משתמשים לא מזוהים/הרשומים –

```
[Authorize]
public ActionResult Create() {...} // Upload new song
```

במוקם שימוש באנווטציות לתייחס מוקומי של הפילטרים, ניתן לבצע רישום גלובלי בرمת האפליקציה כולה מתוך מתודת הרישום הנkirית מהקובץ `Global.asax`: המתודה הסטטistica `App_Start` של המחלקה `FilterConfig` השמורה בתיקייה `RegisterGlobalFilters` שננסקרה לעיל.

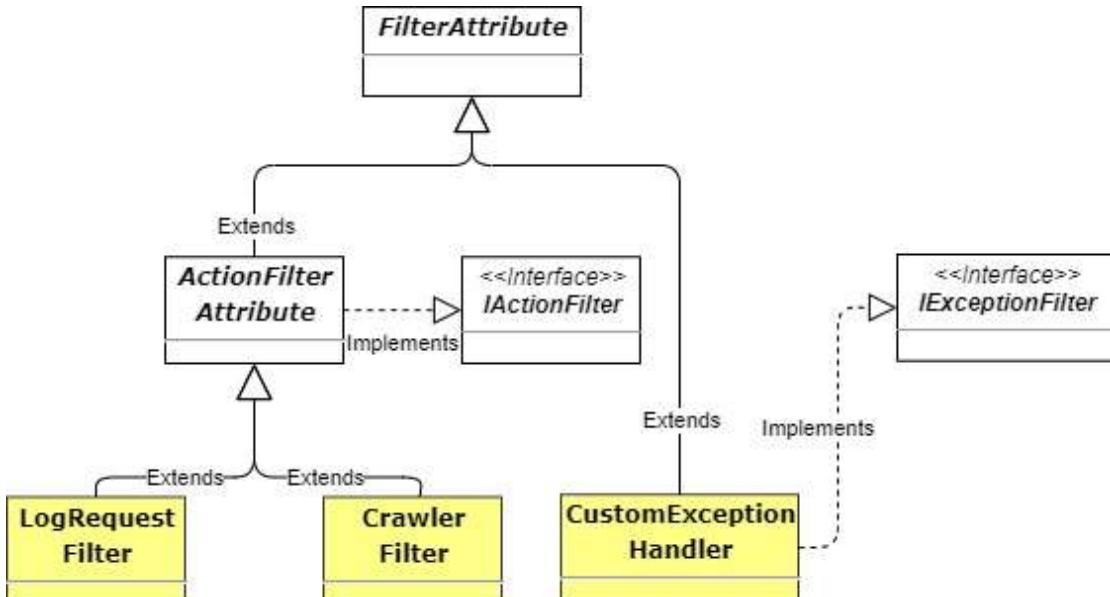
באוטם מקרים בהם יש צורך לזרום או להרחיב את לוגיקת ברירת המחדל המספקת ע"י הפלטפורמה של .NET, علينا להגדיר פילטרים מותאמים אישית לצרכים שלנו (Custom Filters). סעיף זה מתאר את מחלקות ה필טר המותאמות (Custom Filters) שנכתבו לטובת המערכת. מחלקות אלו מגדירות לוגיקה פרטנית שנייה לשbez בנקודות עוגן מוגדרות מראש ש-ASP.NET MVC מימושה. כל המחלקות הללו שמורות בפרויקט בתיקייה `Filters` ובהתאם מוגדרות ב-namespace `CW.Soloist.WebApplication.Filters` –

קוד המקור של כל מחלקות אלו זמין בקישור הבא –

<https://github.com/cwelt/Soloist/tree/master/WebApplication/Filters>

המערכת דורשת/מחייבת את בירית המבדל עם פילטרים מותאמים אישית משני סוגים: פילטרים מסווג Action להתרבויות סבב פועלות של בקרים מסוימים (פירוט בהמשך), ופילטר מסווג Exception לדרישת טיפול בירית המבדל בשגיאות חריגה שלא נתפסו ע"י האפליקציה.

להלן דיאגרמת מחלקה עם מחלקות הפילטר המותאמות שנכתבו לטובת המערכת (אלו המודגשות בצהוב) והקשר שלהם לממשקים שונים ומחלקות-האב האבסטרקטית שהן ירושות ממנה –



7.4.1 פילטר טיפול בחריגות (CustomExceptionHandler)

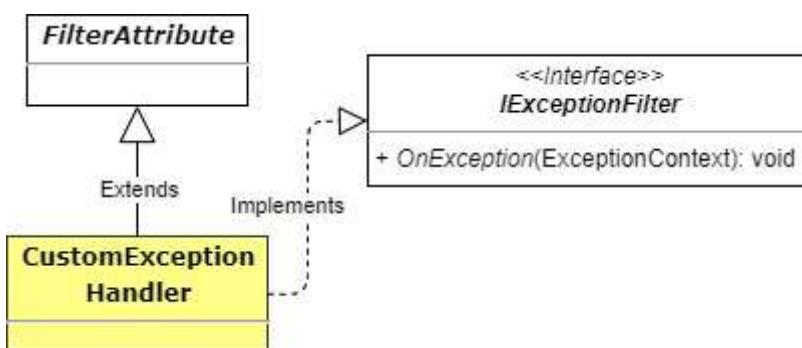
מחלקה CustomExceptionHandler מטפלת בשגיאות Exceptions שלא נתפסו ע"י האפליקציה –

```
public class CustomExceptionHandler : FilterAttribute, IExceptionFilter {...}
```

מחלקת פילטר זו מימושת את המתודה OnException של הממשק IExceptionFilter לטובת החזרת דף HTML מעוצב במקרה של חריגה שלא נתפסה, במקום דף שגיאת בירית המבדל ש-ASP.NET מחזיר. הדף המוחזר נשמר בקונטיקט המועבר כפרמטר. להלן חתימת המתודה –

```
public void OnException(ExceptionContext filterContext) {...}
```

להלן דיאגרמת מחלקה של פילטר CustomExceptionHandler המתארת את הקשר שלו לרכיבי ה필טר הסטנדרטיים בפלטפורמת ASP.NET MVC –

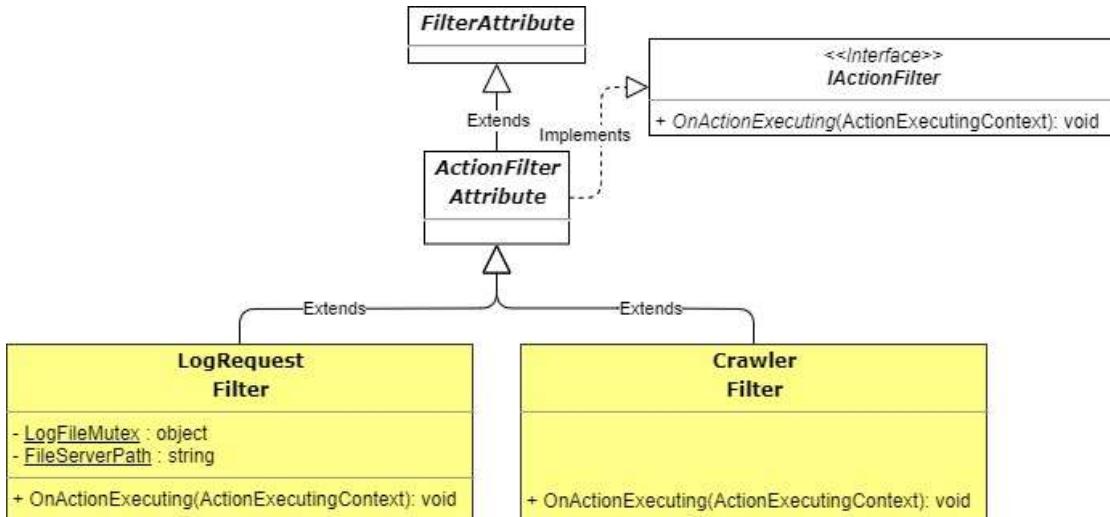


7.4.2 פילטר טיפול בבקשות (ActionFilters)

בתיקייה ActionFilters תחת תיקייה Filters בפרויקט מוגדרות מחלקות פילטר מסווג המגדירות טיפול פרטני סביר בעולות בקרים, המאפשרת להתריבב הן לפני והן אחרי ביצוע הפעולה ע"י הבקר, וכן לפני החזרת התוצאה מפעולה זו ולאחר החזרת התוצאה.

בשלב מומשו שני מחלקות Action Filter עבורי המערכת: הראשונה – CrawlerFilter, האחראית על סינון גישה של בוטים ומונע חיפוש למתחודות מסוימות של הבקרים, והשנייה – LogRequestFilter, האחראית על תיעוד הבקשות השונות מהבקרים בקובץ לוגים לטובת בקרה. שתי מחלקות אלו מתערבות בטרם ביצוע הפעולה ע"י הבקר. לשם כך הן שתייחסן ממושות מתחודה ייחידה OnActionExecuting מהמנשך IActionFilter.

להלן דיאגרמת מחלקה של מחלקות פילטר אלו המתארת את הקשר שלהן לרכיבי ה필טר הסטנדרטיים בפלטפורמת ASP.NET MVC עם המתוודה הרלוונטיות שהן ממושות –



LogRequestFilter 7.4.2.1

```
public class LogRequestFilter : ActionFilterAttribute {...}
```

מחלקה זו מוגדרת לפועל באופן גלובלי באפליקציה באמצעות רישום במתוודה הסטטית RegisterGlobalFilters של המחלקה FilterConfig כפ"י שתואר לעיל. היא רושמת פרטימס כלליים על כל בקשת HTTP המועברת לבקרים ושומרת פרטימס אלו בקובץ לוג על השרת ב-App_Data – Logs. לטובת ניהול הרישום בלוגים המחלקה מגדרה שני שדות: FileServerPath (נקו"ן לעכשו זוהי תיקייה App_Data), מחרוזת המכילה את הנתיב לתיקיית הקבצים של השרת (נקו"ן לעכשו זוהי תיקייה Logs) ו-LogFileMutex – מנעול למניעה גישה במקביל של שני Thread-ים או יותר בעת כתיבה בלוג.

CrawlerFilter 7.4.2.2

```
public class CrawlerFilter: ActionFilterAttribute {...}
```

מחלקה זו מוגדרת באופן מקומי על מתחודות יי'ודיות שעבורן המערכת חסומה גישה לבוטים ומונע חיפוש. לשם כך המתחודות היי'ודיות מסומנות באnotציה מתאימה: [CrawlerFilter]. למשל – מתחודה Contact בבלר דף הבית שמצויה למשתמש את פרטוי יצירת הקשר של מנהלו האטר חסומה בפניהם מונע חיפוש ובוטים על מנת שלא יאחזו מידע השמור השם ויאחסנו אותו במלחכים אוטומטיים –

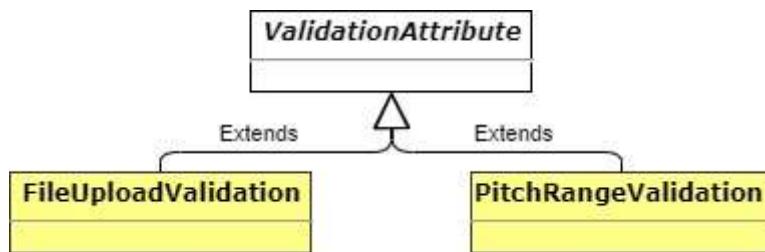
```
[CrawlerFilter]
public ActionResult Contact() => View();
```

7.5 מחלקות ולידציה (Validations)

פלטפורמת ASP.NET MVC מגדרת מחלקות ולידציה לבודיקות תקינות, שאוthon ניתנו להפעיל של שדות בטופס HTML שימושים שלוחים לשרת ע"י סימון השדות המבוקשים לבדיקה באנותציות מתאימות. כל המחלקות הללו יורשות ממחלקת-האב האבסטרקטית `.ValidationAttribute`.

הפלטפורמה מגדרה כבר סט בדיקות תקינות מוכנות מראש, כגון בדיקת תקינות כתובת מייל, בדיקה שערך נומרי נמצא בתחום מסוים, בדיקה ששדה אינו ריק ועוד. כמו כן, ניתן להגדיר מחלקות ולידציה מותאמות אישיות שմגדירות בדיקות תקינות ע"פ הלוגיקה העסקית הנדרשת לאפליקציה. לאחר הגדרת מחלקות אלו, ניתן לסמן את השדות המבוקשים באנותציות המתאימות ובבדיקות אלו ישולבו אוטומטית במנגנון בדיקות התקינות הסטנדרטי ע"י סביבת הריצה של ה-ASP.NET MVC.

סעיף זה מתאר את המחלקות ולידציה השונות שהוגדרו בפרויקט ה-Web עבור המערכת. להלן דיאגרמת מחלקה כללית המתארת מחלקות אלו (בצבע) ואת קשר הירושה שלהם למחלקה-האב האבסטרקטית –



שתי מחלקות אלו מימושות את המתודת האבסטרקטית `IsValid` המתקבלת כקלט אובייקט וكونטיקט רלוונטי, בזקמת את ערך האובייקט ע"פ הלוגיקה המבוקשת בהתאם לתוכן הקונטיקסט ומחזירה `ValidationResult` למנגנון הבדיקה של NET. שבודק את התוצאה ומשיכן את העבודה בהתאם. להלן פירוט מימוש שתי מחלקות הולידציה.

7.5.1 בדיקת תקינות קבצים (FileUploadValidation)

המחלקה `FileUploadValidation` מימושת בדיקות תקינות שונות עבור קבצי הקלט המועלים למערכת – קובץ MIDI וקובץ אקורדים –

```
internal class FileUploadValidation : ValidationAttribute {...}
```

הבדיקה מבוצעת ע"י מימוש המתודת האבסטרקטית `IsValid` שקוראת בתורה למוגדרות עזר פנימיות האחראיות על בדיקות שונות –

```
protected override ValidationResult IsValid(object value, ValidationContext context){...}
```

בדיקות תקינות זו מופעלת ע"י סימון שדות שימושיים להעלות קבצים למערכת עם אנותציה מתאימה של `[FileUploadValidation]`. למשל שדה של קובץ ה-MIDI ב-`SongViewModel`, המודול שמשמש את טופס העלאת שיר חדש –

```
[FileUploadValidation]
public HttpPostedFileBase MidiFileHandler { get; set; }
```

להלן מתודות הולידציה הפנימיות שהמתודה `IsValid` נזורה בהן לטובת מלאכת הולידציה –

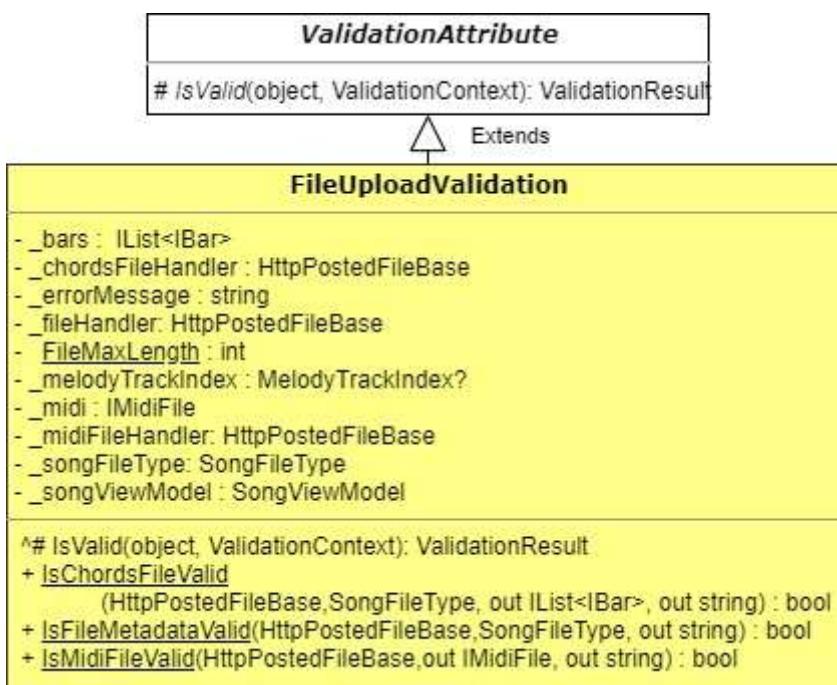
```
public static bool IsFileMetadataValid(HttpPostedFileBase fileHandler, SongFileType fileType, out string errorMessage)
public static bool IsChordsFileValid(HttpPostedFileBase chordsFileHandler, SongFileType fileType, out IList<IBar> bars, out string errorMessage)
public static bool IsMidiFileValid(HttpPostedFileBase midiFileHandler, out IMidiFile midiFile, out string errorMessage)
```

המתודה `IsFileMetadataValid` בודקת נתונים כלליים כגון גודל הקובץ (וידועה שהקובץ אינו חורג מהגודל המקורי המוגדר) וסוג הקובץ (לודא שרק קובצי MIDI וטקסט מועלים ולא איזה סקריפט זמני). המתודה `IsChordsFileValid` מבצעת בודקת שקובץ האקורדים בפורמט המתאים שהוגדר, והמתודה `IsMidiFileValid` בודקת שקובץ ה-MIDI תקין. שתי המתודות האחרונות נזירות בבדיקה שכבר הוגדרו במחלקה [CompositionContext](#).

להלן השדות המוגדרים במחלקה, המשמשים בשימוש חוזר במקום הגדרתם מחדש בכל מתודה בדיקה בנפרד. כמו כן מוגדר משתנה סטטי קבוע להגבלת הגודל המקורי על קובץ המועלה למערכת (נכון לעכשו מוגדר כמגה בית) –

```
private const int FileMaxLength = 1048576; // 1 MB
private string _errorMessage = string.Empty;
private IMidiFile _midi = null;
private IList<IBar> _bars = null;
private SongFileType _songFileType;
private SongViewModel _songViewModel = null;
private HttpPostedFileBase _fileHandler;
private HttpPostedFileBase _midiFileHandler;
private HttpPostedFileBase _chordsFileHandler;
private MelodyTrackIndex? _melodyTrackIndex;
```

להלן דיאגרמת מחלקה של `FileUploadValidation`



7.5.2 בדיקת תקינות טווח צליליים (PitchRangeValidation)

המחלקה PitchRangeValidation מimplements בדיקות תקינות עבור טווח הצליליים שהמשתמש בוחר בטופס הלחנת שיר –

```
public class PitchRangeValidation : ValidationAttribute{...}
```

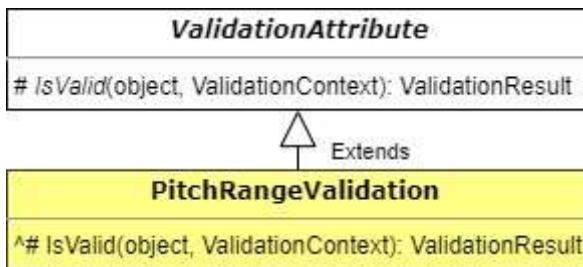
הבדיקה מבוצעת ע"י מימוש המתוודה האבסטרקטית `IsValid` שקוראת בתורה למתוודה הסטטית – `CompositionContext` של המחלקה `IsPitchRangeValid`

```
protected override ValidationResult IsValid(object value, ValidationContext context){...}
```

בדיקות תקינות זו מופעלת ע"י סימונו השדה/שדות שקובעים את טווח את הצליליים ב- ViewModel שנשלח לטופס הלחנה (`CompositionViewModel`) עם אונוטציה מתאימה של – `[PitchRangeValidation]`

```
[PitchRangeValidation]
public NotePitch MaxPitch { get; set; }
```

להלן דיאגרמת מחלקה של PitchRangeValidation



7.6 מודלים לטפסים (ViewModels)

הם מחלקות ViewModels המהוות אוסף שדות (בדומה ל-`struct` בשפת C) ייעודי לטובת איזוחו – View – מסך/טופס במנשך המשמש. בהקשר של מערכת זו, תוצר ה-View הוא דף HTML. מנגנון מנוע ה-Views של ASP.NET MVC משਬץ נתונים בתבניות ה-View המוגדרות. נתונים אלו משובצים מתוך מודל – ViewModel המספק ל-View קלט. להבדיל מהמודלים הנידונים [בשכבה הגישה לנ נתונים](#), ViewModel אין רלוונטיים ל-Persistence. הם נוצרו במיוחד לשירותי ה-Web במנשך המשמש ולשם כך מגדים שדות נוספים לביקורת הרשות והעשרה ממוקש המשמש. סעיף זה מפרט את הגדרות ה-ViewModel השונות. כל ה-ViewModelים מרכזים בפרויקט ה-Web בתיקייה `ViewModels`.

7.6.1 מודל-תצוגה עבור שיר (SongViewModel Class)

המחלקה SongViewModel מגדירה מודל לרכיבו הנתוניים הרלוונטיים למסכים טפסים להציג ועיבוד של יישויות שירים: העלאת שיר חדש, הצגת שיר קיים, עדכון שיר, מחיקת שיר והציג רישימת השירים הקיימים. להלן המאפיינים/שדות המוגדרים במודול זה –

```
public int Id { get; set; }
public string Title { get; set; }
public string Artist { get; set; }
public string Description { get; set; }
public HttpPostedFileBase MidiFileHandler { get; set; }
public string MidiFileName { get; set; }
public string MidiPlaybackFileName { get; set; }
public MelodyTrackIndex? MelodyTrackIndex { get; set; }
public string MelodyTrackIndexDescription { get { ... } }
public HttpPostedFileBase ChordsFileHandler { get; set; }
public string ChordsFileName { get; set; }
public String ChordProgression { get; set; }
public DateTime Created { get; set; }
public DateTime Modified { get; set; }
public bool IsPublic { get; set; }
public bool IsUserAuthorizedToEdit { get; set; }
public bool IsUserAuthorizedToDelete { get; set; }
public bool IsAdminUser { get; set; }
public string StatusMessage { get; set; }
```

המאפיינים מקושטים באנווטציות שונות לבדיקות תקינות והגדרת תיאור ידידותי יותר למשתמש.

המחלקה מגדירה גם שני בנאים: בניית ריק, ובנאי המשמש כמשמעות בנאי העתקה, המקבל קלט מופע של Song, מחלקת מודל שיר, ובונה ממנו את ה-ViewModel –

```
public SongViewModel() { }
public SongViewModel(Song song) { ... }
```

⁴ Plain Old CLR (Common Language Run-time) Object

7.6.2 מודל-תצוגה עבור הלחנת מנגינה (CompositionModel Class)

המחלקה `CompositionViewModel` מגדרה מודל לרכיבו הנתונים הרלוונטיים לטופס הלחנת שיר. להלן המאפיינים המוגדרים במודול זה –

```
public OverallNoteDurationFeel OverallNoteDurationFeel { get; set; }
public MusicalInstrument MusicalInstrument { get; set; }
public NotePitch MinPitch { get; set; }
public NotePitch MaxPitch { get; set; }
public IEnumerable<SelectListItem> PitchSelectList { get; set; }
public CompositionStrategy CompositionStrategy { get; set; }
public bool useExistingMelodyAsSeed { get; set; }
public int SongId { get; set; }
public SelectList SongSelectList { get; set; }
public double SmoothMovement { get; set; } = 15;
public double ExtremeIntervals { get; set; } = 10;
public double PitchVariety { get; set; } = 15;
public double PitchRange { get; set; } = 5;
public double ContourDirection { get; set; } = 5;
public double ContourStability { get; set; } = 10;
public double Syncopation { get; set; } = 10;
public double DensityBalance { get; set; } = 15;
public double AccentedBeats { get; set; } = 15;
public double WeightSum { get; set; } //Sums up all proportional weights
```

המאפיינים המואתולים עם ערך נורמי כברירת מחדל מציגים את המשקל היחסי שיינטן לממד המצווג ע"י אותו מאפיין בשעורך האלגוריתם הגרפי: למשל, המאפיין `SmoothMovement` מייצג את הממד של מעבר חלק בין צלילים שכנים שנאמד ע"י המתודה `EvaluateSmoothMovement` המתווארת [במודול השערון](#). ממד זה מקבל כברירת מחדל משקל של 15% בשעורך הכללי. סך המשקלים צריך להסתכם ל-100%.
המאפיינים מוקשטים באנווטציות שונות לבדיקות תקינות והגדרת תיאור ידידותי יותר למשתמש.

7.6.3 מודל-תצוגה עבור ניהול חשבונות משתמשים

פלטפורמת ASP.NET MVC מגדרה שני קבצי `ViewModel` המרכזיים מחלקות מודל לטובות המסכים של ניהול חשבונות משתמשים – הזדהות משתמשים, שחזור סיסמה ועוד: `ManageViewModels.cs` ו- `AccountViewModels.cs`. תיעוד על מחלקות `ViewModel` אלו ניתן [כאן](#).

7.7 מסכים וטפסים (Views)

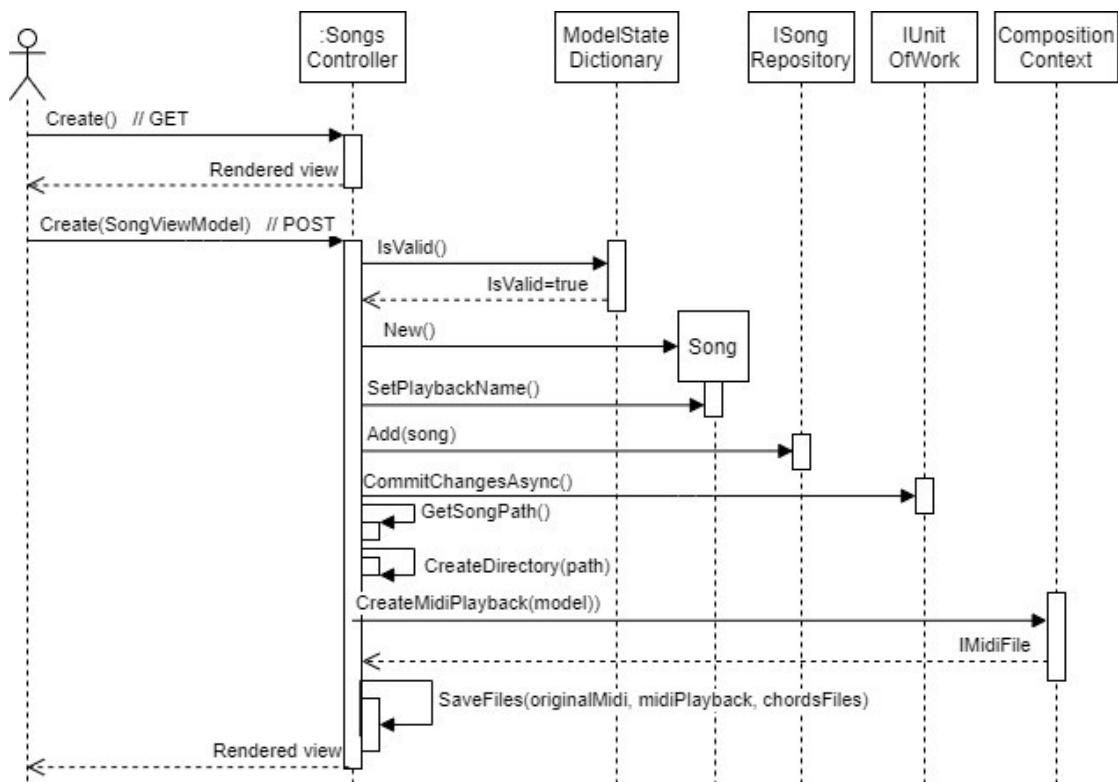
ה-Views בארכיטקטורת MVC הם המרכיבים והטפסים שמוצגים למשתמשים. עבור יישום ה-Web של המערכת, אלו הם למעשה דפי HTML. ע"פ הגדרת המחדל בארכיטקטורה של ASP.NET MVC, כל ה-Views מ羅וצים בתיקיית Views בפרויקט ה-Web, תחת המוסכמה הבאה – כל ה-Views (מסכים, טפסים – דפי HTML) שנשלחים אל המשתמש מתווך בקר מסוים, ושם מ羅וצים בתיקיית Views תחת תת-תיקייה בשם ה-Controller (ללא הסיומת Controller), ובירית המחדל של קובץ ה-View (קובץ Razor עם סיומות cshtml שמנגנון MVC-Create.cshtml ממיר ל-HTML) הוא שם המתוודה של הבקר שמחזירה View זה. למשל, הקובץ View של ASP.NET Views Songs שתיקייתSongs תקינה תבנית המשמשת את מונע ה-Create MVC לבניית דף HTML כאשר משתמשים פונים עם URL מתאים לבקר השירים למתוודה (מנקודת המבט של המשתמש, כאשר מבוצע ניווט למסך העלאת שיר חדש באפליקציה).

בשל הקשר החזוק בין ה-Views לבין מותודות הבקרים שמנוטות אליהן, פשר ומהות ה-Views השוניים הם כפויים למתחודות הרלוונטיות בסעיף [הברקים](#). למען הסדר הטוב, להלן תקציר תיקיות ה-Views השונים המוגדרים במערכת (התוכן עצמו של תיקיות אלו הוא ה-Views המתאימים למתחודות שהתייעד שלhn מפורט בסעיף [Controllers](#) – Controllers-Sub-Folder).

Views Sub-Folder	Description
Account	ASP.NET MVC Predefined views for assisting the management of user AAA (Authentication, Authorization and Accounting).
Manage	
Composition	HTML form for composition a new melody for an existing song which is already stored in the system.
Home	Views that are returned from actions that are managed by the home page controller such as about page, contact page, etc.
Shared	General view templates and html pages such as a standard 404 "not found" error page, navigation bar configuration, and a default html layout for all views.
Songs	Screen and forms that are managed by the Song controller for uploading, querying and updating songs.

7.8 אינטראקציה בין רכיבי ה-MVC

סעיף זה מציג דיאגרמת רצף (Sequence diagrams) עבור אחד התהליכיים המרכזיים במערכת – להמחשת –



8 דפוסי עיצוב

סעיף זה מתאר את דפוסי העיצוב (Design patterns) המרכזיים שהוטמעו בפיתוח המערכת.

8.1 דפוסי יצרה (Creational Patterns)

8.1.1 דפוס מפעל (Factory Pattern)

אחד מעקרונות הפיתוח המוביילים שהוטמעו בפיתוח המערכת הוא –

"Program to an Interface, not an Implementation"

בהתאם, כמעט לכל מחלקה פונקציונאלית במערכת מוגדר מנשך כל שמחצין ומגדר באופן אבסטרקטiy את הפונקציונליות המוגדרת עליה (הכוונה במחלקה פונקציונאלית היא למחלקה מגדרה מותודות ולא משתמש רק כמבנה המקבץ אוסף מאפיינים). לאחר החצנת פונקציונליות זו, ניתן להחליף אצל הקליינטים את כל הكريאות המבוצעות למחלקות הקונקרטיות בקריאות למוגדרות המשקפים, כך ש מבחינת הקליינטים האינטראקציה תהיה מול ה-Interface בלבד ולא במחלקה קונקרטית. טכניקת תכנות זו מאפשרת להחליף בעתיד שימוש קיים של ה-Interface במימוש אחר באופן ርመעט ושקוּף לקלינייטים, שמיילא עובדים מול ה-Interface ולא מול איזושהי מחלקה קונקרטית, וכך עד מהלכה החדשת ממלאת את דרישות ה-Factory. קלינייט זה ርመעט ושקוּף. למה ርመעט שקוּף לחלוּתוֹ? כאן נכנס דפוס ה-Factory לתמונה.

אפשרו אם כל הكريאות למוגדרות מצד הקלייניט מבוצעות מול Interface ולא מול מחלקות קונקרטיות, בכך לקבל איזושהו מופע של מחלקה הממשת את ה-Interface, כלומר, ככלומר, באיזושהו שלב חייבות להיות בקוד איזושהו קריאה לבנאי של מחלקה קונקרטית הממשת את המשן. אם קריאה כזו נמצאת בצד הקלייניטים, אז עדין יש תלות במחלקה קונקרטית: בכל פעם שיידרש להחליף את מימוש, יש למצוא את כל הكريאות לבנאים של אותה מחלקה ולהחליף אותן בكريאות לבנאים של המחלקה החדשה, בתקופה שיש תאימות כללית בפרמטרים של הבנאים מהמחלקות השונות. אמנס עדין שיטת עבודה זו עדין חוסכת את הצורך גם את הكريאות למוגדרות אולם הינו רוצים שחרור טוטאלי מהתלות של קלינייטים במחלקות קונקרטיות. לשם כך אנחנו נזירים בדפוס ה-Factory: אנחנו מחזינים את האחריות לייצור מופעים של מחלקות קונקרטיות לאיזושהו גורם צד ג' (the-Factory): הקלייניטים יפנו אליו בבקשת לקבלת מופע של איזושהו Interface, והוא יהיה אחראי להחזיר מופע של איזושהו מחלקה קונקרטית הממשת Interface זה, באופן שקוּף לקלינייטים. באופן זה, כל הكريאות לבנאים מוכזות במקום אחד, וכך שאמ בעtid יהיה צורך בהחלפת המחלקה הממשת, כל השינוי יהיה מבודד למקום אחד בלבד – ה-Factory, ומהצד של הקלייניטים הדבר יהיה שקוּף לחלוּתוֹ.

המערכת אינה מימושת את דפוסי ה-Factory הקלאסיים "by the book" כפי שתוארם בספר של GoF – Gang Of Four, אלא בטכניקה שיתור דומה ל-"Simple Factory" המוצגת בספר DP של Freeman בHead First. הסיבה למימוש זה היא שבמסגרת המערכת לא עלה הצורך לנחל משפחה של אובייקטים מסוימו מפועל, ולכן לא היה צורך ב-Abstract Factory, וכן לא היה עניין בירושה מחלוקת מפועל צו או אחרת ושינוי המימוש שלא להחזרת מופעים אלטרנטיביים, וכך גם לא היה צורך ב-Factory Method.

לפיכך המימושים שהוגדרו הם בעיקר בתצורה של מחלוקת מפעל סטטית המגדירה מוגדרות סטטיות (סוג של Factory Methods) היוצרות מופע של איזושהו Interface ייוזדי. בחלק מהמפעלים המופע המוחזר תלוי בבקשת הקלייניט, למשל ב-ComposerFactory המקביל כארגוומנט אסטרטגיית הלחנה מבקשת ובהתאם יוצר מופע של Composer קונקרטי מתאים, ואילו בחלק מהמפעלים מוחזר פשוט מופע ישיר של המחלקה היחידה הממשת את ה-Interface: במקרה כל עניין ה-Factory הוא לבצע deference של הקריאה לבנאי לטובה שבירת התלות של הקלייניטים במחלקה הקונקרטית.

בפרויקט Web לעתות זאת הוטמע שימוש בספרייה Autofac המנהלת תלויות של קלינייטים בממשקים ב-Dependency Injection (IoC Container), ודואגת להזניק תלויות אלו בטכניקת Inversion of Control בזמן ריצה ע"י יצירת מופעים ב-Factory פנימי של הספרייה על-סמן Container קונפיגורציות מוגדרות מראש של יחסי התלות בין מנשכים והמחלקות הממשות אותם.

לහן רכיבי ה-Factory שהווטמעו במערכת –

- .1 - מחלקה המגדירה מתודות יצירה עבור מנשקי היישיות המוסיקליות השונות – משך-שהיה (INote), תו (IDuration), אקורד (IChord) ותיבה (IBar).

- .2 - מחלקה המגדירה מתודות יצירה עבור מופעים של המנשך IMidiFile המייצג קובץ MIDI.

- .3 - מחלקה המגדירה מתודות יצירה עבור מופעים קונקרטיים של המחלקה האבסטרקטית Composer בהתאם לסטרטגיית הלחנה מבוקשת המוגדרת כערך ב-Enum CompositionStrategy של מוגדרת למוגדרת למתודה המפעל ארגומנט.

- .4 - בפרויקט ה-Web בקובץ ההגדרות глובאליות Autofac Dependency Injection, במסגרת המתודה Application_Start במחלקה Global.asax.cs רישומי תלויות ב-IoC המנוהל ע"י ספרייה Autofac כלהלן –

```
// create an autofac inversion of control container
ContainerBuilder builder = new ContainerBuilder();

...

// register dependencies
builder.RegisterType<EFUnitOfWork>().As<IUnitOfWork>().InstancePerRequest();
builder.RegisterType<SongRepository>().As<ISongRepostiory>().InstancePerRequest();
```

משמעות ההגדרות שלעיל, הוא שזמן ריצה, תחת הקונטסט המוגדר, עבר כל תלות של קליניטים במנשך IUnitOfWork, ספריית Autofac תציג לשחקן מופיע של המחלקה הקונקרטית EFUnitOfWork, ובאותו אופן, עבר כל תלות במנשך ISongRepostiory, ספריית Autofac תציג מופיע של המחלקה הקונקרטית SongRepository. לפיכך, תוך שימוש בספרייה זו בפרויקט ה-Web, הבקרים (Controllers) לא צריכים להכיר את המחלקות הקונקרטיות בשכבה הגישה לנוטונים (DAL). כל שהם צריכים זה לעבוד מול ה-Interfaces של ניהול הישיות (האינטגרטור IUnitOfWork) שאחראי על האינטראקציה מול ה-DB ושל מאגרי הישיות שנדרש לאחזר ו/או לעדכן IRepository (לצורך העניין). ספריית Autofac מגדירה שימוש משלחה לדפוס ה-Factory ומחייבת מופעים מתאימים באופן שקווי לקליינט של יישום ה-Web, כך שבקשר הזה אין לו מושג האם מדובר ב-EntityFramework או חבילת ORM אחרת או אולי אפלוי שימוש ישיר ב-API של ADO.NET.

למעשה, ניתן היה (ואולי אף מוטב היה) לעשות שימוש רוחבי בספרייה Autofac בכל הפרויקטים במערכת ולהחסוך את הצורך בהגדרת מפורשת ידנית של מחלקות ה-Factory, שכן ספרייה זו מספקת פונקציונליות נוספת נספנת כגון קביעת-h Scope של המופעים (למשל מופיע יחיד עבור כלל הבקשות – Singleton, או מופיע חדש לכל בקשה – Prototype), ריכזו כלל התלוויות בمكان אחד מרכז בمكانם במחלקות Factory שונות, ואולי הכי חשוב: ממצעת Dependency Injection אצל הקליניטים וחוסכת מילכתחילה את הצורך בפניה מפורשת ל-Factories.

עם זאת, הפרויקטים השונים במערכת פותחו בזמןים שונים: חלק מה-Factories במחלקות הלוגיקה העסקית הוגדרו טרם הגדרת פרויקט ה-Web בשכבה התצוגה, כך שהטמעת ה-Autofac בדיעבד מצריכה עבודה נוספת נוספת, מה שגם שהקונפיגורציה בפרויקט Class Library של שכבה הלוגיקה העסקית שונה שמווגדרת מפרויקט Web של ASP.NET MVC. נקודת נוספת נוספת לטובת התצורה הקיימת בשכבה הלוגיקה העסקית היא שהגדירות דרך מחלקות ה-Factory Dependencies מօסף הגדרת Dependencies באיוזה קובץ קונפיגורציה שמעלים את הרבה יותר ברורות, להבדיל מօסף הגדרת Dependencies באיוזה קובץ קונפיגורציה שמעלים את כל הלוגיקה ונראה כמו "קסם" מבלי שברור לבדוק איך הדברים עובדים מחורי הקלעים, ומתי ואיפה נוצרים המופעים ומה scope שליהם.

8.2 דפוסים מבניים (Structural Patterns)

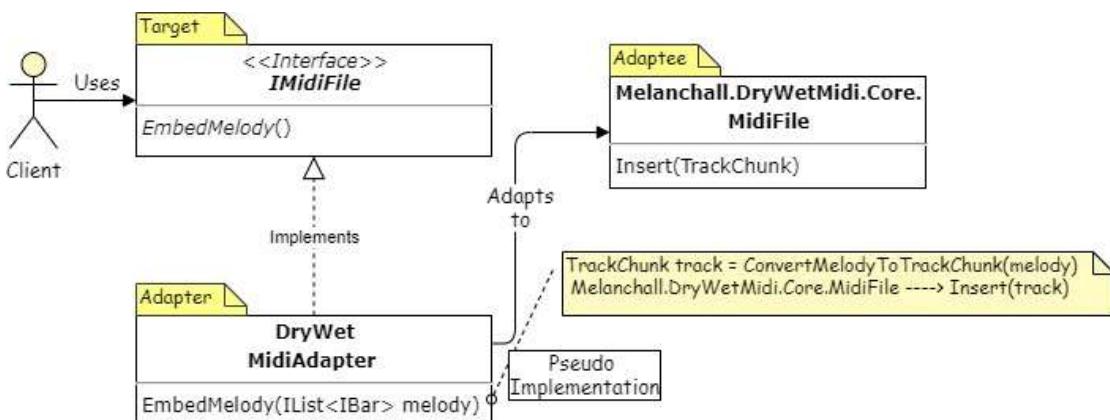
8.2.1 דפוס מעטפת מתאמת (Adapter Pattern)

דפוס ה-Adapter יושם במערכת **בתת-שכבה שירותית ה-MIDI**: פלטפורמת ה-.NET. אינה מספקת ספריות סטנדרטיות ב-C# לתמיכה ואינטראקציה עם פרוטוקול MIDI. המערכת צריכה שירותים בסיסיים כגון – קרייה וכ כתיבה של קובץ MIDI, נגינת קובץ והחלפת רצועות בקובץ. לטובות – DryWetMIDI שירותים אלו נעשו שימוש בספרייה חיצונית בשם DryWetMIDI <https://github.com/melanchall/drywetmidi>

ספרייה זו עוטפה את הפרטים הטכניים (low-level) של ה프וטוקול ומספקת מנשך פשוט יחסית לעובדה מול קובצי MIDI ב-high-level, אולם גם מנשך זה דורש לא מעט עיבוד והתקומות כדי שנitin יהיה לעובד מולו במערכת. למשל: בשילול להטמעה מגינה חדשה שהולחנה כרצועה בקובץ MIDI, יש להמיר את כל התווים כפי שמיצגים במערכת (ע"פ התיאור בסעיף **יצוג ישויות מוסיקליות**) לארווי MIDI כפי שמיוצגים ע"י הספרייה החיצונית. זה כולל המרות בזנים של משכי שהיא, בטיפוסים, שמות ועוד. ספריות אחרות העובdot מול פרוטוקול MIDI עושות לספק מנשך אחר לממרי לקליינטים. לפיכך, היה חשוב למנווע צימוד (coupling) בין קוד הלביה של המערכת לבין הקוד של הספרייה החיצונית, ולעבוד מול אבסטרקציה (IMidiFile interface) לצורך העניין), כך שאם בעתיד עליה הצורך להחליף הספרייה הקיימת באחרת, הדבר לא צריך שינוי אצל הקליינטים כל עוד הספרייה החדשה תדאג לעמוד בתנאי האבstraction שהוגדרו.

דפוס ה-Adapter מתאים מאד עבור תרשים זה: המנשך של המערכת שמייצג קובץ IMidiFile הוא על תקן ה-target, Target Interface, המחלקה MidiFile מהספרייה החיצונית היא על תקן ה-Adaptee, והמחלקה DryWetMidiAdapter היא על תקן ה-Adapter. הקליינטים הם תחת השכבות/שכבות אחרות העושם שימוש במנשך הנח IMidiFile, דוגמת מול אבסטרקציה CompositionContext בשכבה הלוגיקה העסקית, ובקר ה-CompositionController בשכבה התצוגה בפרויקט ה-.ASP.NET MVC

להלן תרשימים מחלוקת הממחיש שימוש טיפוסי ב-Adapter: קלינט מבקש ממנשך היעד IMidiFile implements מומחה המשמש בקובץ MIDI. מחלוקת ה-Adapter (DryWetMidiAdapter) ראהית ממירה את להטמעה מגינה בקובץ MIDI. מחלוקת ה-Adaptee (Melanchall.DryWetMidi.Core.MidiFile) היא המנגינה מהייצוג הפנימי לייצוג החיצוני של ספריית ה-Adaptee, הלא היא מנגינה Melanchall.DryWetMidi. לאחר ההמרה, מחלוקת ה-Adapter יכולה לעשות שימוש ישיר במנשך Shmachka.h.melanchall.DryWetMidi. Adaptee מחשקת ה-Adaptee מהספרייה החיצונית מספקת, עובדה עם Collection של רצועות פשוטות להוציא את הרצועה החדשה (למען פשטות השימוש פרטם אחדים כגון פרטן אינדקס) –



הערות: זהה כמבנה גרסת המימוש של object adapter (ולא class adapter) (class adapter, שכן שפת C# (להבדיל מ- C++ למשל), אינה תומכת בירושה מרובה. כמו כן, מעבר לאדפטציה למחלוקת הספרייה החיצונית MIDI של הספרייה החיצונית, ה-Adapter עוטף שירותים נוספים של הספרייה הנתונים ב-low level יחסית, ומנגיש אותם ב-high-level, וכך מתפרק גם קצר C-Facade לקליניטים.

8.3 דפוסים התנהגותיים (Behavioral Patterns)

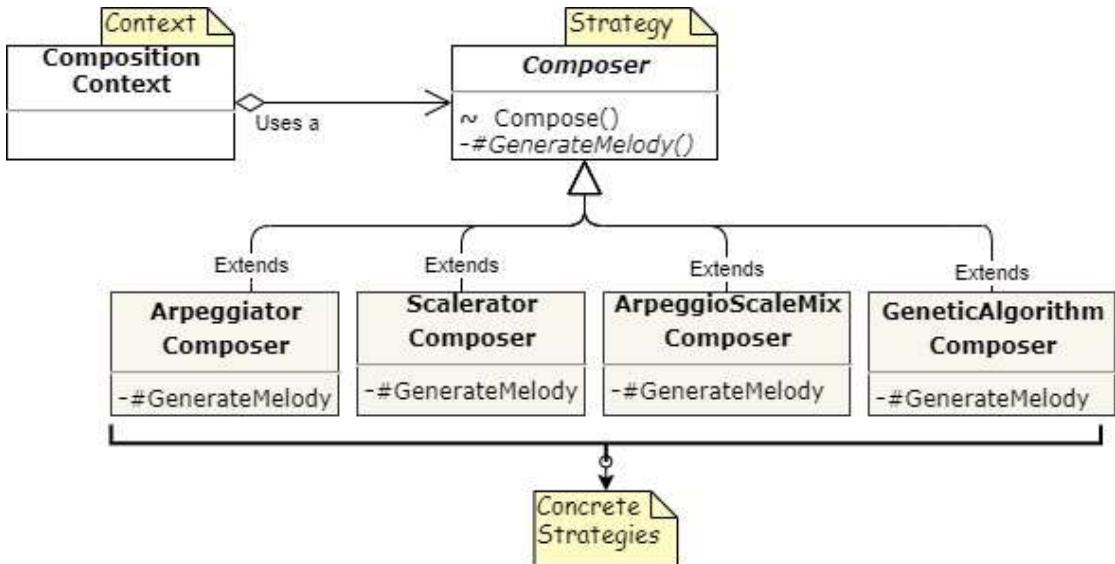
8.3.1 דפוס אסטרטגיה אלגוריתמית (Strategy Pattern)

האלגוריתם המרכזי שהמערכת משתמש בו להלנת מנוגנות הוא האלגוריתם הגנטי המתוואר בסעיף לחנה עם אלגוריתם גנטי. עם זאת, אין זה האלגוריתם היחיד המשמש באוטומציה וכוח עיבוד לטובת הלחנה. בתחום ב-Algorithmic Composition קיימים מגוון אלגוריתמים ושיטות נוספות לדוגמה, כגון אלגוריתם המבוסס על חוקי-דקדוק ושפה, אלגוריתמים מבוססים מערכות-מושחה, ואלגוריתמים המבוססים על מודלים מתמטיים כגון שימוש בתהליכי סטוכסטיים אי-דטרמיניסטיים.

אין זה מן הנמנע שהאלגוריתם הגנטי יוחלף באלגוריתם אחר, בכך לא לשכתב את כל המערכת מחדש עבור כל אלגוריתם, המערכת עצמה מראש הפרדה לתתי-שכבות עבור הייצוג, טיפול בקובצי ה-MIDI, ואלגוריתם הלחנה.

בכך לאפשר החלפה דינמית של אלגוריתם הלחנה, הוטמע דפוס ה-Strategy : מחלקה האב-האבסטרקטית Composer מגדירה אחת חיצונית Compose שאותה הקלינינטים מכיריים, ומגודלה נוספת – GenerateMelody – פרטית ואסטרטקטית, הנקרהת מתוך Compose, שאחריותה לממש את הלחנה בפועל. מחלקות קונקרטיות ממושאות מוגדרת זו עם אלגוריתם הלחנה המבוקש, ובבחינת הקונטקט שմבקש לבצע את הלחנה, זה שקווי לו, הוא תמיד קורה לאוינה מוגדרת Compose מאיזשהו מופיע של Composer ללא מודעות מהי המחלקה הקונקרטית שמבצעת את הלחנה בפועל.

להלן תרשימים מחלקה המתאר את רכיבי המערכת המשתתפים בהטמעת דפוס ה-Strategy:



מטרים פשוטים, הדיאגרמה מסתירה פרטיים מימוש טכניים מסוימים: לטובת הלחנה דינמית בזמן ריצה של אלגוריתם הלחנה המבוקש, המערכת מגדרה אוסף קבועים המייצגים את אלגוריתמי הלחנה השונים המוגדרים במערכת ב-Enum בשם CompositionStrategy. קונטקט השילוב מוגדר במתודת הלחנה פרטיר אופציונלי שבו קלינינטים יכוילים לציין את אסטרטגיית הלחנה המבוקשת ע"י ציוו ערך ה-enum המתאים. (אם לא מצוין אף ערך, המערכת בוחרת כברירת מחדל את האלגוריתם הגנטי). קונטקט הלחנה פונה עם אסטרטגיית הלחנה המבוקשת ופונה למחלקה מפעלי יצירה של מלחינים (ComposerFactory) ומבקש ממנו להחזיר לו מפעע של Composer בהתאם לאסטרטגייה המבוקשת. המפעע מחזיר מופע של מחלקה קונקרטי המממש את Composer, Compose ו-Composer משמש בה להלחנה באמצעות מוגדרות Compose. דפוס עיצוב זה הופך את הקוד לגנרי לחלוtin ביחס לאלגוריתם הלחנה: בשום שלב לא קלינינטים ולא לklass קונטקט הלחנה יש שמאץ של מושג מהי המחלקה הקונקרטיבית המוציאה לפעול את מלאכת הלחנה. כל העבודה של הקלינינטים נעשית מול הקונטקט, וכל העבודה של הקונטקט מובוצעת מול הממשק המוגדר ע"י המחלקה האבסטרקטית Composer. כך שאם בעידן תוגדר מחלקה נוספת היורשת מ-Composer, וויגדר עבורה ערך מתאים ב-Enum המותאם לאסטרטגיות, קלינינטים יוכל לציין ערך זה, ולקונטקט זה יהיה שקוף לחלוtin!

8.3.2 דפוס תבנית אלגוריתמית (TemplateMethod Pattern)

כאמור, אלגוריתם החלטה המרכזី במערכת הוא האלגוריתם הגנטי. למעשה אין זה מדויק לומר אלגוריתם הגנטי עם 'הא הידיעה': אלגוריתם גנטי הוא שם כולל למשפחה של אלגוריתמים, אשר המשותף להם הוא שימוש בשיטות חיפוש היוריסטיות תוך התבססות עמוקה על עקרונות ותובנות מתחלים אבולוציוניים המתרחשים בטבע. אלגוריתמים גנטיים שונים עשויים להכיל מימושים שונים לחלווטין, פירוט נוסף על כך ניתן למצאו בעבודת הסמינר של המחבר בנושא –

<https://github.com/cwelt/Soloist/blob/master/Design/Documents/pdf/Genetic-Algorithms-For-Melody-Generation-Seminar-Paper.pdf>

עם זאת, על אף שמדובר במשפחה של אלגוריתמים, יש בספרות המדעית העוסקת בנושא מוסכמת לגבי איזושהי תבנית כללית של אלגוריתם גנטי טיפוסי. להלן לצד האלגוריתם –

1. אתה תזק זיך זאען
2. אַגְּזָעָן
2.1 אַנְפֵּסִיָּה
2.2 אַוְּגָזִיָּה
2.3 שִׁיאַלְאָמָן
2.4 גַּמְיכִימָּת זיך גַּאֲזָק
3. אַזְּקָק הַאֲמָט אַמְּקִיְּם תְּרָזָאִי הַגְּזִיכָה
3.1 אַטְּמָן, הַמְּצָלָה כְּפָגָן אֶת הַאֲזָאָגָן(אַמְּטָר) הַכְּבִיל(אַמְּטָר) קַיְמָת וְסִינְאָן;
3.2 אַחְלָתָן, חַזְוֹן <i>סְפֵּגָטָה</i>

מימושים שונים של לצד כליה זה יכולים לבוא לידי ביטוי בכל אחד ואחד מן השלבים של האלגוריתם. להלן כמה סוגיות לדוגמה, אשר כל אחת מהן עשויה להוביל למימוש שונה לעומת גמריו –

- **אתחול:** האם לבצע אתחול מכלון על-סמך ידע מוקדם, או אתחול אקראי לחלווטין ?
- **שחלוף:** איזו שיטת שחלוף לבצע? כיצד לבחור את המועמדים מהאוכלוסייה שישתתפו בשלב השחלוף? כיצד לקבוע את מספר נקודות השחלוף והמייקום שלחן?
- **מוטציה:** איזה מוטציות לבצע? האם להגדיר מוטציות ניטרליות לגמרי, או האם המוטציות צריכות להיות מודעות למצוות והתוכן של המועמדים? כיצד לקבוע את המועמדים שייעברו מוטציה? האם ובאיזה תנאים עדיף לבצע שינויים נרחבים על-פני שינויים מזעריים?
- **שיעורץ:** מהי הדרך הטובה ביותר לאמוד את טיב-aicoot הפתורנות המועמדים? כיצד יש להתמודד עם tradeoffs בין מדדי הצלחה מנוגדים?
- **בחירה דוח המשך:** איזה חלק ייחס מהאוכלוסייה יש לسان/לבחרו? האם לבחור תמיד את המועמדים הטובים ביותר, או לתת הזדמנויות גם למועמדים אחרים בעלי פוטנציאל? האם לבחור תמיד רק מועמדים מהדור האחרון או גם מועמדים מדורמים?
- **תנאי עצירה:** כיצד לקבוע متى לעצור? האם ניתן להעריך איזשהו ממד על מגמת התכנסות של האלגוריתם? האם לעצור אחרי מספר קבוע של איטרציות? האם יש איזשהו סף איקות שאם מגיעים אליו ניתן לעצור ?

החלטות שונות בסוגיות אלו מובילות להתפלויות שונות לגמרי ומגדירים מימושים שונים שיכולים להביא לידי פלטים שונים לגמרי. בכך לספק תמייהה בשלד האלגוריתם הטיפוסי ויחד עם זאת לאפשר גמישות השימוש בשלבים השונים של האלגוריתם, המחלקה `GeneticAlgorithmComposer` בצד לאפשר מחלקות לרשות ממנה ולממש אותן באופן אלטרנטיבי. מימוש זה אינו המימוש הקלאסי ("by the book") של דפוס `TemplateMethod` : המחלקה `GeneticAlgorithmComposer` אינה אבסטרקטית, אלה כבר מספקת מימושי ברירת מחדל לכל שלבים האלגוריתם, עם זאת, לכל אחד משלבים אלו היא מספקת מספר מימושים אלטרנטיביים, כך שניתן בקלות להרכיב מימושים שונים ע"י ירושא ממחלה זו ושינוי הקומבינציה של המתוודות הפנימיות שמתוודות ניהול מפעילות – למשל להחליף במתודה `SelectNextGeneration` את הקריאה ל-`PlusSelection`. בקריאה ל-`RouletteWheelSelection`.

8.4 tabniot-odposim-architektutonim

מעבר לדפוסי העיצוב המתוארים בספר של GoF, מוטמעים במערכות דפוסים נוספים המתוארים בספרו של Martin Fowler בשם Patterns of Enterprise Application Architecture, המתאר דפוסים המסייעים בעיצוב ארכיטקטורה חכמה של ישומים ומערכות גדולות רחבות היקף:
<https://bit.ly/34Einlr>

8.4.1 model-tzoga-bkra (MVC)

שכבת התצוגה של המערכת הוגדרה בפרויקט Web בפלטפורמת h-ASP.NET MVC. שם הפלטפורמה מرمזת על שם הדפוס שעליו היא מושתתת: **MVC** – Model-View-Controller. הדפוס כבר מוגדר במסגרת הפלטפורמה, וכל שנוצר הוא להיצמד אליו –

- **Models** הרלוונטיים ל-Persistence הועברו מפרויקט h-Web לפרויקט h-DAL, כך שניתן יהיה לעשות בהם שימוש חוזר גם בפרויקטים אחרים (למשל אם יהיה צורך בכך ניתן יהיה לשימוש במודלים אלו בפרויקט WPF המפותח בתבנית העיצובית של MVVM. מעבר למודלים אלו, הוגדרו בפרויקט h-Web מודלים ייעודיים לריכוז נתונים רלוונטיים לתצוגה, כגון תאים ידידותיים למשתמש של קבאים, רשימות ערכים לבחירה ונתוני הרשאה. מודלים אלו רוכזו בתיקיה ייעודית **ViewModels**. המודלים מכילים רק לוגיקה סיבב אחיזה הנתונים למודל ללא לוגיקה עסקית. המודלים אינם מודעים לא לבקרים ולא ל-VIEWS.
 - **Views** מגדרים תבניות המורכבות מ-HTML, CSS, JavaScript ו-ASP.NET, ומנווע של NET. אחראי לפרש התבניות אלו וליצור מהם דפי HTML לתצוגה לאחר שהוא משbez בהם נתונים מהמודלים המועברים ל-Views. Views מגדירים איזה Model הם מצפים לקבל, אך אינם מודעים לבקר ואינם מכילים לוגיקה למעט לוגיקה הרלוונטית לאופן התצוגה (למשל החלטה אם להציג הודעה ממכשיר צבע או דום).
 - **Controllers** ומשמשים כנתבים או אינטגרטורים, מקבלים בקשה ממשתמשי הקצה, אוספים נתונים רלוונטיים למודלים, מעבירים את המודלים ל-View המתאים ומחזירים למשתמשי הקצה את View המעובד עם הנתונים המשובצים מהמודל (או לחולפן מחזירים ערך אחר במקום View דוגמת מסמך PDF / קובץ MIDI בהתחם לאותה הבקשה).
- פירוט נוסף על שימוש הפרויקט ניתן למצוא בסעיף שכבת התצוגה. אינטראקציה לדוגמה בין ה-View וה-Controller ניתן לראות בסעיף אינטראקציה בין רכבי ה-MVC.

8.4.2 miyopi-avbiyikti-relatsionni (ORM)

שכבת הגישה לנוטונים ממומשת מאחוריה הקלעים באמצעות ספריית EntityFramework. ספרייה זו מimplements את תבנית ה-(ORM) Object-Relational-Mapping, האחראית על גישור הפערים בין ייצוג היישויות העסקיות במודל הקונספטואלי של עולם האובייקטים בפרדיגמת OOP, לבין ייצוגם במודל הפיזי בעולם MSDI-הנתונים הרלציוניים. למשל – בעולם ה-OOP יש יחסי ירושה, ויאלו בטבלאות ב-DB אין. קיימים גם פערים בתיפוסי שדות (לעומת string או varchar,nvarchar), ייצוג משתנים בוליאניים, אפשר/אי-אפשר ערכי null לעמודותNomoriotot ו-etc). פער נוסף הוא נדרש בטבלה קשר ב-DB עברו מידת קשר של "Many-to-many", כאשר בעולם האובייקטים ניתן פשוט להגדיר במחלקה אוסף יישויות מהטיפוס של המחלקה השנייה ללא "מחלקת קשר".

כלי ORM ובפרט ספריית EntityFramework מספקים מענה להרבה מבעיות אלו, ע"י המרות אוטומטית בין הייצוגים של שני המודלים (הكونספטואלי מול הפיזי), ומספקים מנשך נח המאפשר לקליניטים ניהול אינטראקציה מול ה-DB הגרפי בשפת OOP טהורה בטרמינולוגיה ובסמנטיקה של היישויות בעולם האובייקטים עם מחלקות ו��ודות ולא שימוש ב-SQL. שימוש זה חוסך לא מעט עבודה, וופט את הקוד ממשמעותית.

פירוט נוסף על אופן הטעמאת כלי ה-ORM של EntityFramework במערכת ניתן למצוא בפרק מייפוי בין אובייקטים לרשומות (EntityFramework ORM)

8.4.3 הפצת אינטראקטיבית מול DB (Repository & Unit Of Work)

דפוס ה-**Repository** משמש כמאגר זמן ריצה (מנוחל זיכרון ה-RAM) של ישויות, לרוב מאותו טיפוס, המספק מעין מנשך מיילוני (כגון CRUD – Create, Read, Update, Delete) לאחזר ועדכו ישות המנוהלות במאגר זה, תוך הסתרת האינטראקטיבית מול מדיום האחסון הפיסי שבו הישות מאוחסנת בפועל.

דפוס ה-**Unit Of Work** משמש לניהול עדכונים מटבערים המבוצעים באיזשהו Session בזמן ריצה אל מול ה-**Database** שעדיין לא נרשם פיזית: רישום פיסי (ביבוצע Commit) על כל שינוי בפרט יגרור מחיר כבד בביבוצים, ולכן עדיף לנחל עדכונים ב-Batch-ים, אולם אז יש את התקורה של ניהול היסטוריית השינויים מאז ה-**Commit** האחרון תוך מודעות לחשיבות סדר העדכונים (למשל אם מדובר בעדכון של רשומה חדשה, יש להקפיד לבצע את פקודת ההכנות של INSERT טרם פקודת העדכון UPDATE). זהו תפקידו של ה-**Unit of Work**, לנחל מעקב אחר כל הישויות הרלוונטיות לו, לרשות מי עודכן ומתי, לאסוף את כל השינויים האלו המתבצעים באיזושהי יחידת זמן / טרנסאקטיבית ולבסוף לפנות אל מדיום האחסון הפיסי ולבצע את הרישום (Commit) בפועל.

בגוזל, ספריית ה-**Entity Framework** שתוואר בסעיף קודם, מימושה דפוס ה-**ORM** קומבינציה של דפוסי ה-**Repository** וה-**Unit of Work** באמצעות המחלקה **DbContext** ומאפייני ה-**DbSet< TEntity >** מתחם התיעוד של

– (<https://bit.ly/34GfD75>) Microsoft

A DbContext instance represents a combination of the Unit Of Work and Repository patterns such that it can be used to query from a database and group together changes that will then be written back to the store as a unit. DbContext is conceptually similar to ObjectContext.

לפיכך המימוש קיים "בגוזל" (לפחות איזושהי קומבינציה שלו). עם זאת, שימוש ישיר בישויות ה-**DbContext** וה-**DbSet** יוצר תלות ישירה של הקלייניטים (במקרה הזה הקלייניט הוא שכבת הגישה לנתונים) בספרייה ה-**Entity Framework**. אם בעתיד נרצה להחליף ספרייה זו בכלים ORM אחרים, דוגמה NHibernate או אפילו החלפה לעובודה ישירה מול ה-**API** של NET, הדבר יצריך פחות או יותר שכתב מחדש שכבת הגישה לנתונים לאחר מכן יהיה מעורער ותלו依 ב-API של ה-**Entity Framework**.

ב כדי לשבור תלות זו, הוגדרה רמת הפצת **נוספת**, מעלה ה-**Entity Framework**, המגדירה מנשקים עבור שני הדפוסים שלמעלה – **IRepository** (עבור דפוס ה-**Repository**) ו-**IUnitOfWork** (עבור דפוס ה-**Unit Of Work**), ושכבת הגישה לנתונים עובדת רק מול המנשקים האבסטרקטיים, ואני מודעתת למימוש הקונקרטי מהורי הקלעים.

בפועל, המימוש מבוצע כਮובן באמצעות **Entity Framework**, שכאמור כבר מספקים תמייה בדפוסים אלו, لكن כל מה שנדרש הוא להגדיר מחלקות חוצצות שפשוות לVOKE את הביקשות של הקלייניטים ומעבירה אותן להלאה ב-**delegation** למחלקות הרלוונטיות ב-**DbContext**: הידועות לעשות את העבודה: מבצע פעולות של **IUnitOfWork** כגון שמירת השינויים ב-**DB**, וה-**DbSet**-ס שלו מבצעים פעולות של ה-**Repository** כמו אחזור רשות ישיות מה-**DB**, חיישן רשומה לפי מפתח (או פרדייקט אחר כלשהו) ומחיקת ישיות מהמאגר.

במובן זה, מחלקות חוצצות אלו מזכירות קצת את דפוסי ה-**Adapter** ו-**Proxy**. פירוט נוסף על מימוש זה ניתן למצוא בפרק **שכבת הגישה לנתונים (DAL)**, בתתי-פרקאים של **מנשכים** ו-**ניהול יחידות עבודה (Repositories)**.

9 תיאור שינויים עתידיים

סעיף זה מכיל תיאור של מספר שינויים עתידיים אפשריים במערכת והסביר על הטעמanton. נחלק את השינויים לשתי קטגוריות – האחת: שינויים תשתיתיים – הקשורים לפלטפורמה וסביבות ומערכות חיצוניתות שהמערכת תלואה בהן כדי לרווח, אך אינם קשורים לפונקציונאליות או הלוגיקה העסקית של המערכת, והשנייה: שינויים אפליקטיבים/פונקציונאליים – שינויים במערכת עצמה, דהיינו בלוגיקה העסקית המוגדרת בכדי לספק פונקציונאליות אחרת.

9.1 שינויים תשתיתיים

9.1.1 שינויים תשתיתיים הקשורים ל-DB

9.1.1.1 החלפת מסד נתונים

שרת ה-DB הנוכחי המשמש את המערכת הוא שרת MS-SQL Server chinmi, שמספק שטח אחסון מוגבל. נניח שיהיה עניין להחליפו בשרת DB המסופק בשירותי ענן של Azure למשל, שאولي יגבו עלות קטנה, אבל יספקו שירותים נוספים ושטח אחסון נרחב.

אם השרת שם מספקים גם הוא של SQL Server מבית Microsoft, כל שיידרש הוא הסבת תוכן DB-הקיים אל ה-DB החדש (העתקת הסכמה ותוכן כל הטבלאות), ועדכו פרטי החיבור ב-App.config, Connection String בקובץ הקונפיגורציה (קובץ ה-XML של ה-Web.config וכדי) בהתאם לשרת ה-DB החדש.

אם השרת ה-DB הוא שרת אחר, למשל שרת MySQL מבית Oracle או שרת PostgreSQL או שרת Oracle מבית Microsoft, כל שיידרש בנוסך לצדים שלעיל, בהנחה שקיים מחלקה Data Provider מתאימה ב-.NET. (אנלוגי ל-JDBC Driver ב-Java, או JDBC Driver ב-.NET). הוא גם ייובא ה-Assemblies של ספרי נתונים זה, ציוו ה-namespaces שלו ועדכו הפרטים שלו בקובץ הקונפיגורציה ב-XML-ים שליעיל במקומות פרטי SQL Server הספרים של נתונים.

9.1.1.2 החלפת טכנולוגיית ORM

המערכת משתמשת בספרייה Entity Framework גרסה 6 שלה, בתור טכנולוגיית ה-ORM המבוצעת את המיפוי, המרות וגיור בין הרשומות השתוות בטבלאות ב-database למודל הקונספטואלי. גרסה 6 נחשבת כיום גרסה ישנה: פלטפורמת ה-.NET Framework. נמצאת היום במגמת שינוי ומיורציה לטכנולוגיות Cross-Application שאינן תלויות בפלטפורמה או מערכת הפעלה (להבדיל למשל מטכנולוגיות Windows שפותחו על-ידי WPF שモוגלת אך ורק לישומי-.NET Core,.NET Framework ועוד). טכנולוגיות אלו מפותחות תחת פלטפורמת ה-.NET Core,.NET Framework העדכנית ביותר, בעבר טרם שחרורה יוחס לה גרסה מספר 7, נכתבו מחדש כדי להתאים ל-.NET Core,.NET Core ויצאה תחת השם Entity Framework Core.

טבע יהיה לשדרג לגרסה העדכנית ביותר, או לפחות את הטכנולוגיה הנוכחית הנחשבת כישנה, באחת אחרה, דוגמת NHibernate. לאחר שליבת שכבת הגישה לנוטונים מוגדרת לעובדה אך ורק מול מנשכים אבסטרקטיים (IRepository ו-UnitOfWork), אין תלות בטכנולוגיית Entity Framework, כך שבדי להחליף את טכנולוגיית ה-ORM הקיימת, ככל שיידרש הוא כתיבת מחלקות מתאימות למימוש המנשכים האבסטרקטיים שליל תוך delegation וأدפקציה מתאימה למחלקות הטכנולוגיה החדשה, ועדכו הקונפיגורציה של הזරקת התלויות (Dependency Injection) של ספריית Autofac בקובץ Global.asax.cs כך שה-Factory של הספרייה יחזיר מופעים של המחלקות החדשות במקום אלו הקיימות.

9.2 שינויים בפלטפורמה

שכבה התצוגה הנוכחית של המערכת ממומשת בפרויקט Web בפלטפורמת ASP.NET MVC אשר מושתת על אינטראקטיבית. אחד היתרונות של יישומי Web שהוא נגיש מכל פלטפורמה (בהתאם יש לכתוב את הקוד בצד client כך שיהיה מותאם לממדים המכשירים השונים, למשל תוך שימוש נכוון בספריות Bootstrap).

אולם, נניח שהוא צריך גם בפיתוח אפליקציה native לאיזשהו מכשיר, למשל כאפליקציה natvie לאנדרואיד ו/או OS. ובכן, לאור החלוקה של המערכת לשכבות, עיקר המאמץ יהיה פיתוח מנשך המשתמש (GUI) ל-smartphone שיחליף את שכבת התצוגה הקיימת. כל הלוגיקה העסוקה המתווכמת של האלגוריתם הגנטי, ייצוגו תווים ושימוש ב-DB, כל אלו יותרו ללא שינוי. הם רק שירותים ששכבת התצוגה יכולה לצרוך ואינם תלויים בклиינט שגורץ את השירותים הללו. נניח למשל שנפתחים אפליקציית בטכנולוגיות Xamarin, אזי שכבת התצוגה שגדירים שם יכולה לצרוך את שירותי הحلחנה של המערכת תוך שימוש במחלקה `CompositionContext`, ואת שירותי ה-Database לשימירה וzechzer של שיריים קיימים תוך שימוש בשכבת הגישה לנוטונים.

אם יש צורך ביצירת השירותים ע"י אפליקציה הכתובה בטכנולוגיה אחרת לגמרי, למשל אפליקציה הכתובה ב-python המיעודת לרוץ על מערכות הפעלה של Linux, עדין ניתן יהיה ביכולת להסביר את השירותים הולכים ע"י עיטופם ב-API (למשל בטכנולוגיית ASP.NET Web API 2 או ASP.NET.Core), ואז בנוסף לאספקת שירותי אלו למשתמשי קצה באמצעות דפי HTML באתר אינטראקטיבי, המערכת תספק גם end-points: נתיבי URL שנייתן לפניות אליהם ותקבל את השירות ב-API, שאנו תלוי פלטפורמה.

9.2 שינויים אפליקטיביים/פונקציונאליים

9.2.1 הוספת אלגוריתם הלחנה

המערכת מגדרה כת אלגוריתם מרכזי יחיד – האלגוריתם הגנטי. אמנים מוגדרים מספר מחלקות נוספות במקביל למחלקה האלגוריתם הגנטי, אולם הן די מנוונות: האלגוריתם שלהם פשוט מחולל רצפי צלילים בסיסיים על מנת צלילי האקורדים והטולמות המתאימים למחזור ההרמוני של השיר ללא יישום לוגיקה נוספת.

נניח שנרצה לשלב במערכת אלגוריתם נוסף שעשויה לעבוד יסודית בדומה לאלגוריתם הגנטי, למשל איזה אלגוריתם Machine Learning שמנתח מאגר יצירות לדוגמה הנינתנות לו כקלט וnochshavot כ-"טובות" ומבחן מנגינה בהשראה ובסגנון של יצירות אלו. כבودה של מרכיבות האלגוריתם במקומה, אבל בהינתן אלגוריתם שכזה, ככלمر בהנחה שכבר קיימים מימוש של האלגוריתם, העובד על ישויות מסויקליות ע"פ היצוג המוגדרת במערכת `IDuration`, `INote` וכו'), כל שיידרש לשלב את האלגוריתם במערכת הוא להגדיר מחלקה חדשה היורשת מהמחלקה האבסטרקטית `Composer` לשבץ את מימוש האלגוריתם החדש במסגרת המודול האבסטרקטית `.GenerateMelody`.

במידה ויידרשו לאלגוריתם החדש פרמטרים נוספים שאינם מועברים כיום למחלקות ה-`Composer` הקיימות – למשל איזשהו מבנה נתונים מורכב או לכל היותר נתיב או URL למאגר היצירות שעל האלגוריתם לנתח, ניתן לעשות שימוש בפרמטר `customParams` המוגדר במתודה `Compose` (שהינה המתודה שהקלינייטים מפעילים לקבלת השירות). פרמטר זה הינו מטיפוס מערך באורך בלתי מוגבל של `object[] customParams`, כך שניתן להעביר בו כל פרמטר נוסף שהוא ולשבץ אותו במשתני המחלקה במסגרת המתודה `InitializeCompositionParams`. המקלט פרמטר זה מהמתודה `Compose`.

שינוי זה מתאפשר הודות למימוש דפוס העיצוב `Strategy`.

9.2.2 עדכון האלגוריתם הגנטי הקיימים

נניח שנדרשה להכניס שינויים באלגוריתם הגנטי הקיימים. נציג שינוי רצינלי שטבעי שייהה בו צורך בעtid.

9.2.2.1 קיבוע צלילים בנקודות עוגן

בثور נגן גיטרה חשמלית חובב, לא פעם באלאטור או חיבור מגינה חדשה יש לי ריעון כללי על איזשーン נקודות עוגן בмагינה: הצליל הראשון שבו אני מעוניין לפתח את המגינה (או סדרת הצלילים), הצליל האחרון שאנגן ב כדי לסייע את המגינה, ועוד מספר נקודותicia שליא מקומות. אח"כ אני מנסה לחבר את הקשרים שבין הנקודות, ככלומר למצוא איך להגיע מהצליל המתנגן בנקודות הפתיחה לציליל בתקנת הביניים הבאה באופן המשמש לקבע נקודות עוגן לשימוש ו/או לשדר. לפיכך היתי רוצה שהמערכת תאפשר לי בثور משתמש בקבוע נקודותicia שליא מבחן גובה צליל ו/או משך השהייה, שימושה לי שיישארו כמות שהן במאגרה החדש שתיווצר, ככלומר האלגוריתם רשאי להחליף את כל הצלילים שבין הנקודות, אך באותו נקודותicia מוגדרות התווים המקוריים יישמרו.

בצד נוכל למש Feature שכזה במסגרת האלגוריתם הגנטי הקיימים?

הודות לתצורה הנוכחית של האלגוריתם, המגדירה שלד כללי המפעיל מתודות ניהול המוגדרות `virtual`, וחלוקת למודולים נפרדים האחראים למש מתודות אלו, ניתן בקבות חסית לשנות ו/או להרחיב את הלוגיקה הקימית של האלגוריתם תוך שימוש בדף `Template Method`: נגידר מחלוקת חדשה היורשת מהמחלקה `GeneticAlgorithmComposer`, והיא תוכל לבצע `override` למתודות הרלוונטיות של האלגוריתם הגנטי, לרבות מתודות האתחול הראשית `InitializeCompositionParams` במחalkerת-האב `InitializeCompositionParams` כמתודה `virtual`, כך שגם אותה ניתן לדודס במחלקות יורשות ע"פ הצורך.

לאחר הגדרת מחלוקת היורשת מ-`GeneticAlgorithmComposer` השאלה היא איפה המקום hei טוב "להתערב".

ראשית, ב כדי לציין את מיקומי נקודות העוגן, ניתן להיעזר בפרמטר הכללי `[params object customParams]` המועבר למחלוקת במסגרת המתודה `InitializeCompositionParams`. לאחר שפרמטר זה מקבל מערך בכל אורך סופי שנבחר של משתנים מכל טיפוס שנבחר, יוכל להגיד תצורה להעברת מיקומי נקודות העוגן – למשל איזשהו עץ שהצמתים ברמה הראשונה שלו מייצגים אינדקסים לתיבות ברצף התיבות, והצמתים ברמה השנייה מייצגים אינדקסים של התווים שיש לקבע בתוך אוסף התווים השיך לTİיבת המוצגת ע"י צומת האב ברמה שמעל. תצורה זו של אוף יי'צוג נקודות העוגן וממבנה הנתונים שישמש לכך – אלו הם פרטיטים טכניםים שכבודם מונח במקומות, אך הם לא משפיקים על הארכיטקטורה והמשך שילוב השינוי המבוקש במערכת, שכן לא ניתן בהם עוד, אלא רק נניח שהפרמטר `customParams` מכיל את נקודות העוגן, כך שהמחלקה החדשה היורשת מ-`GeneticAlgorithmComposer` יכולה לאחסן נקודות אלו כראוניה ויש לה גישה לנקודות אלו בכל שלב באלגוריתם.

שנית, אחרי שיש לנו את הקלט הרלוונטי, יש לבחון כיצד ואיפה להתייחס אליו במסגרת התצורה הקימית. דרך אחת, היא לעבור במחלקה היורשת על כל המתודות שמדוונות מנגינות – מתודות אתחול, מתודות מוטציה ומודות שחלוף ולעדיין אותן באופן שיתיחס לנקודות העוגן וישמר עליהם מפני כל שינוי. דרך זו נשמעת ארכוכה, מסורבלת ומוגעת לפורעניות (למשל באגים הנובעים מפספוסים של מגוון קומבינציות ומקרים קצה). דרך אלטרנטיבית, עצלית, קצחה ופושטה יותר, היא כן לאפשר לאלגוריתם לשנות את כל הצלילים לרבות נקודות העוגן כפי שהיא עד כה, ובסיום כל מחזור (אייטרציה) של האלגוריתם, להריץ מתודה מוטציה חדשה שתוגדר במחלקה החדשה העוברת על אוכלוסיות המנגינות החדשנות ומטמיה בהן את הערכים המבוקשים בנקודות העוגן. לחיפוי, ניתן לספר ביצועים ולבצע הטמעה זו רק בסיום כל המחוירים בסוף האלגוריתם בטרם החזרת הפלט הסופי לקליניטים, אולם אז שלב השערוך יהיה פחות מדויק שכן הוא ייקח בחשבון את המנגינות שמיילא עתידות להשתנות טיפה לאחר הטמעת נקודות העוגן, וכן עדייף לבצע זאת בכל מחזור בנפרד לאחר שלב המוטציה ולפניה שלב השערוך. ניתן למשל לעשות זאת מתוך מתודה ניהול המוטציות `Mutate`, לאחר ריצת כל יתר המוטציות.

10 נספח: רקע תאורטי של המוסיקה המערבית

סעיף זה סוקר בקצרה מונחים בסיסיים מהטרמינולוגיה המוסיקלית שמוצרים במסגרת הסמינר.

1. שנים עשר הצלילים של הסולם הכרומטי

גובה צליל (pitch) מוגדר ע"י תדירות גלי הקול של הצליל – ככל שהתדר של הצליל גבוה יותר, כך הוא גם נשמע גבוה יותר (ולהפך). בהמשך נזכיר ונאמר פשוט 'צלילי' כאשר הכוונה היא לגובה צליל. **מרווח** הוא היחס שבין שני (גביה) צלילים. מקובל למדוד יחס זה ביחסות של טוון (או חצאי-טוון). **חצאי-טוון** הוא המרווח הקטן ביותר הנמצא בשימוש במוזיקה הטונאלית המערבית, והוא מבטא יחס של $\frac{1}{2}$ בין תדריות גלי הקול שני צלילים הסמוכים זה לזה (דהיינו צלילים הנמצאים במרחק של חצאי-טוון זה מזוה). בתרבויות אחרות (למשל במוזיקה היהודית ובמוזיקה ערבית) משתמשים אף במרווחים קטנים יותר של רביעי-טוונים.

מרווח של שנים-עשר חצאי טוונים נקרא **אוקטבה**. בהתאם ליחס בין תדריות גלי קול המוגדר ע"י מרוח של חצאי-טוון, מרוח האוקטבה מגדר יחס של $1:2^{12}$: $1:2^{12} = 1:4096$. דהיינו תדריות כל צליל היא בדיק פ-שתיים מהתדרות הצליל הנמוך ממנו באוקטבה. צלילים הנמצאים במרווח של אוקטבה בדיק זה מזה דומים מאוד אחד לשני, עד כדי כך שבמוזיקה המערבית מסווגים אותם לאוטו צליל בסיסי. בהתאם לכך, מבחנים בין סה"כ שנים-עשר צלילים בסיסיים, כך שכאשר מתייחסים לאיזשהו צליל בסיסי, הכוונה היא בעצם המשפחה או מחלקה של צלילים, המכילה איזשהו צליל מסוים ואת כל הצלילים הנבדלים ממנו בכפולות של שלמות של אוקטבה.

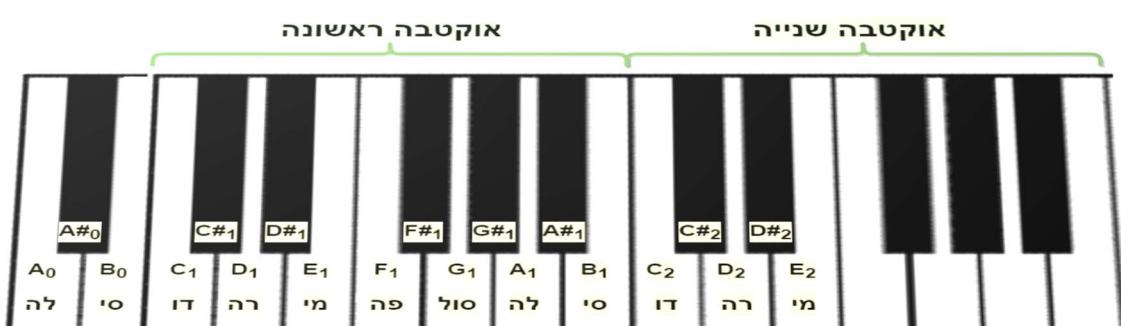
את שנים-עשר הצלילים הבסיסיים נהוג לסמן ע"י שבעת הסמלים לטיניים (לה, סי, דו, רה, מא, פה וסול) ו**סימני התק' #** (דייאז) ו-**ט'** (במול) המציינים שניוי של חצאי-טוון כלפי מעלה אומטה, בהתאם. לפיכך את שנים-עשר הצלילים הבסיסיים נוכל לרשום באופן שקול כך –

A, A#, B, C, C#, D, D#, E, F, F#, G, G#

או כך –

A, Bb, B, C, Db, D, Eb, E, F, Gb, G, Ab

(ובהתאם: לה, לה-דייאז, סי, דו, דו-דייאז,...,סול-דייאז או לה, סי-במול, סי, דו, רה-במול,...,לה-במול). בכך להתייחס לאיזשהו גובה (תדר) קונקרטי של אחד משנים-עשר הצלילים הבסיסיים, מה שמכונה גם גובהו האבסולוטי, ונוהגים לחלק את כל הצלילים לסדרות של שנים-עשר צלילים עוקבים, החל מ-**C** (דו), ולסדרן על-פי מיקומן היחסי מנמוך לגבוה: בפנסטר סטנדרטי בעל שמנונים ושמונה קלידים, סדרת שנים-עשר הצלילים המופקים ע"י שנים-עשר הקלידים התואמים המתחילה מקליד ה-C השמאלי ביותר במקלדת (שהוא גם ה-C הנמוך ביותר במקלדת) נקראת מקלדים הראשונה. הסדרה העוקבת לה מימיינן נקראת האוקטבה השנייה, וכן הלאה. באופן זה, ניתן לסמן גובה-צליל קונקרטי ע"י ציון הסמל המייצג את הצליל הבסיסי, סימן התק' במידה נדרש, ואינדקס המציין את מספר האוקטבה הרלוונטי, למשל – C1 מייצג את גובה הצליל C הנמוך ביותר במקלדת הפנסטר, C2 את גובה הצליל C הגובה ממו באוקטבה אחת וכן הלאה, להלן המחשה של תווויות שמות/סמלים הצלילים המופקים ע"י 20 הקלידים הראשונים של מקלדת פנסטר סטנדרטית בעלת 88 קלידים –



מנעד הוא היקף של איזשהו טווח של צלילים. לפיכך מנעד של מקלדת פנסטר סטנדרטית הוא 88 חצאי טוון.

עדכון האלגוריתם הגנטי הקיים

שינויים אפליקטיביים/פונקציונאלים

2. תווים

בכדי לifyיג ו/או לתאר יצירה מוסיקלית באופן גרافي על הכתב/דפוס, משתמשים בתווים. **תווים** הם סמלים המתארים גובה צליל או הפסקה (שקט) וכן את משך שהיא הצליל/הפסקה, אשר נקרא גם הערך הרитמי של הצליל/הפסקה. ברישום ניתן לציין גם תוכנות הקשורות לצורה המיועדת להפקת הצליל, כמו דרגות דינאמיקה (חזק/הדגשה) ותחושים מהירות/זרימה הכללית. התווים עצם רשומים על גבי **חמשה**, מערכת של חמישה קוים אופקיים מקבילים.

2.1. גובה צליל

גובה צליל מיוצג ע"י גובה יחסיתו של רישום הצליל ביחס למפתח מסוים, עם סימן התקן מתאים ע"פ הצורך. להלן דוגמה של רישום הצלילים החל מ-A3 (לה באוקטבה החמישית) ועד ל-G#5 (סול דיאז באוקטבה החמישית) על חמשה לפי מפתח סול⁵:

2.2. תיבות, קצב, משקל וערך ריטמי (משך שהייה קבוע)

התווים בחמשה מחולקים **لتיבות**, לרוב באורך קבוע, המהוות ייחדות לוגיות המפרידות בין קבוצות של תווים. ניתן למספר את התיבות ולעתוף אותן בסימני חזרה והוראות בקרה המנוחות לחזור עליהם ו/או לעבור למקטע מבוקש (באופן דומה להוראות מחשב כמו repeat ו- (goto).

יחידת הזמן הבסיסית במוזיקה היא **פעימה**. קצב הפעימות מתואר בד"כ ע"י ציון ערך BPM⁶, המציין את מספר הפעימות שיש בדקה אחת. למשל, BPM של 120 מציין כי הקצב הוא שתי פעימות בכל שנייה.

לכל יצירה מגדריים **משקל** קבוע ע"י שבר $\frac{p}{q}$: המכנה q מציין את מכנה יחידת הזמן הבסיסית המיצגת פעימה בודדת שהיא $\frac{1}{q}$, והמונה p מציין כמה ייחדות זמן בסיסיות יכולים יש בתיבה. למשל, המשקל הנפוץ $\frac{4}{4}$ מציין שיחידת הזמן הבסיסית לפעימה היא רביע (ערך המכנה הוא ארבע) וכן שיש בכל תיבה ארבעה רביעים (ערך המונה הוא ארבע). בכדי לציין את **משך שהייה** שלתו נתנו, מה שנקרא **ערך הריטמי**, מתאימים לתו סימוני קצב המתאים לאורך השהייה שלו ביחס לתיבה שלמה. למשל, במשקל של $\frac{4}{4}$,תו עם ערך ריטמי שלם מתmesh לארוך תיבה אחת שלמה (ארבעה פעימות), חצי מתmesh לאורך חצי תיבה (שתי פעימות), רביע למשך רביע תיבה (פעימה אחת בלבד), וכן הלאה. להלן המחשה של סימונים של כמה ממשכי הצליל הבסיסיים במשקל $\frac{4}{4}$

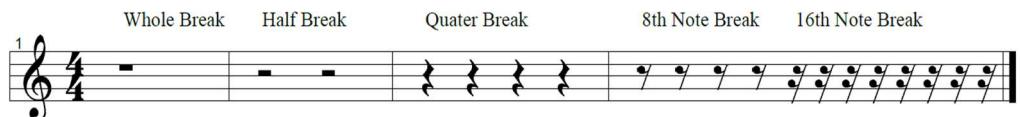
⁵ מפתח סול מציין שמייקום התו G4 (סול באוקטבה רביעית) על פני החמשה הוא השורה השנייה מלמטה בחמשה.

⁶ BPM = Beats Per Minute

ambilי להיכנס לפרטי אופן הסימון, נציין כי המכנה של יחידת זמן משך שהיית צליל אינו מוגבל דווקא לכפולות של שתיים: ישנן דרכים רישום לכל מספר רצionario. כמו כן, תיבה לא חייבת להיות הומוגנית ביחידות אורך, ניתן לשלב בה יחידות אורך מעורבות כל עוד משך השהייה הכלול של כלל התווים יחד בתיבה תואם למשקל הקצב היותר ואינו קצר ממנו או חורג ממנו.

כמו כן, ניתן להאריך את משך השהייה צליל/תו אל מעבר ליחידת הזמן הבסיסית שלו ע"י רישום מופע נוסף שלו העוקב למופע המבוקש וקיים שני המופעים בקשת חיבור. הארכה שכזו קובעת את משך השהייה הכלול של התו לסכום משכי הזמן של התווים המוחברים בקשת. למשל, ניתן לצוין יחידת זמן מנוקדת, שימושה היא משך של פי 1.5 מיחידת הזמן המקורי ללא הניקוד. ע"י שימוש בשתי הטכניקות לעיל ניתן להרכיב אינסוף ערכיים ריאתמיים שונים, וליצור **סינкопות**, הדגשה של איזשהו تو ע"י ניגונו שלא על הפעימה עצמה, למשל ע"י קיצור משך השהייה של התו שקדם לו/nochti/, וניצול הזמן שהתפנה כדי להקדים טיפה את הנגינה של התו הנוכחי, פעולה שמעוררת אפקט של הפתעה וענין.

עבור הפסקות (שהיות של שקט) בין צלילים יש סימונים קצב יייעודיים, להלן הסימונים של הפסקות ביחידות זמן הבסיסיות מעל משלך 4/4 :



3. הרמונייה ומלודיה

באופן כללי, **מלודיה** (מנגינה) מוגדרת ע"י רצף של צלילים המונגנים אחד אחרי השני (לחוד), והרמונייה מוגדרת ע"י רצף של צלילים המונגנים יחדיו (במקביל). ההרמונייה מלואה את המlodih.

4. סוגי מרוחחים

כאמור, מרוחחים במוסיקה המערבית מרוחחים נבדדים ביחידות של טוֹן (וחצאי טוֹן). היחס הנקבע ע"י מרוחחים מסוימים משרה איזושהי תחושה כאשר המrhoוחה מנוגן באופן הרמוני (זהינו כאשר שני הצלילים היוצרים את המrhoוחה מנוגנים יחדיו במקביל). תחושה זה יכולה להיות מסוגת כנעימה (קונסוננטית), צורמת (דיסוננטית) או זכה (צלולה, טבעית). בהתאם, המrhoוחחים מסווגים לכמה קבוצות מתאימות. להלן רשימה של המrhoוחחים הכלולים באוקטבה אחת –

מחלקה	מרווח בطنאים	מרווח
מרוחחים דיסוננטיים	חצ'י טוֹן	סקונדיה קטנה
מרוחחים דיסוננטיים	טוֹן	סקונדיה גדולה
מרוחחים קונסוננטיים	טוֹן וחצ'י	טרצה קטנה
מרוחחים קונסוננטיים	שני טוֹנים	טרצה גדולה
מרוחחים זכימ'ים	שניים וחצ'י טוֹנים	קוורטה
מרוחחים דיסוננטיים	שלושה טוֹנים	טריטון
מרוחחים זכימ'ים	שלושה וחצ'י טוֹנים	קויניטה
מרוחחים קונסוננטיים	ארבעה טוֹנים	סקסטה קטנה
מרוחחים קונסוננטיים	ארבעה וחצ'י טוֹנים	סקסטה גדולה
מרוחחים דיסוננטיים	חמישה טוֹנים	ספטימה קטנה
מרוחחים דיסוננטיים	חמישה וחצ'י טוֹנים	ספטימה גדולה
מרוחחים זכימ'ים	שישה טוֹנים (שניים-עשר חצ'י טוֹנים)	אוקטבה

הערה : המrhoוחחים נבדדים הוא גובה הצליל האבסולוטי (חצ'אי טוֹנים) והוא בסימול התוווי ע"י המרחק בין התווים בסולם הכרומטי, למשל המרחק האבסולוטי בין דו-דייז לבין מי-במול הוא טוֹן, אולם לאחר שבסולם הכרומטי של שנים-עשר הצלילים, יש בין דו למי את התו רה, המrhoוח הזה של טוֹן לא יסועג כסקונדיה גדולה, אלא כטרצה קטנה מוקטנת.

5. סולמות, טונאליות ודרגות הרמוניות

סולם הוא סדרה מוחזרת של מרווחים, הקובעת איזשהו יחס סדר בין צלילים, שמרחיקם זה מזה מוכתב ע"י סדרת המרווחים של הסולם. במוסיקה הטונאלית המערבית, בכל סולם שכזה יש צליל אחד שהוא "השליט" בסולם, והוא נשמע הכי טוב וטבעי בקונטקט רצף הצלילים שבסולם: הצלילים בסולם נמצאים באינטראקציה הדדית תמידי של יצירה מתח ופתורנו ע"י רגעה, והרגעה או אתנהתא הגדולה ביותר של הרצפים בסולם שוואפים אליה היא כאשר מתגנש הצליל ה-"שליט" של הסולם. צליל זה מכונה ה-"טונייה". צליל הטונייה משתמש גם כשורש הסולם, דהיינו כצליל שמננו מתחילה את סדר רצף המרווחים, שקובע בהתאם את הרכב הצלילים בסולם.

לאוסף הצלילים בסולם מיחסים דרגות, ע"פ מיקומם היחסי ברצף הצלילים שבסולם ביחס לטונייה:תו הטונייה עצמו הוא דרגה ראשונה, התו שרחוק ממנו במרקח הנקבע ע"י המרווח הראשון בסדרת המרווחים של הסולם הוא דרגה שנייה וכל הלאה. על בסיס דרגות אלו בונים אקורדים, רצפי שלושה צוים שונים או יותר המנגנים יחדיו. כל סולם מכתיב אקורדים הנגזרים מהדרגות שלו כלהלן: לכל דרגה (תו) בסולם מסויפים את התווים שנמצאים ממנו במרקח של 3,5 ו-7, במובנים של דרגות. לאקורדים הנוצרות באופן זו קוראים **דרגות הרמוניות**. להלן דוגמה של הדרגות הרמוניות הנוצרות מהסולם דו-מיזור, הנקבע ע"י סדרת המרווחים: טון-טון, חצי טון, טון, טון, חצי טון, שמושרש בתו דו (כלומר, סדרת המרווחים קובעת סולם מז'ור כלשהו, ואשר משרישים סדרה זו בדו בתור צלילי הטונייה, מתקבל הסולם דו-מז'ור) –

I - Cmaj7	II - D minor 7	III - E Minor 7	IV - F maj7	V - G7	VI - A minor	VII - B minor7 b5

לפיכך, כל סולם מכתיב אוסף צלילים וакורדים. ישנו סולמות שונות בעלי תתי-קבוצות משותפות של צלילים, لكن בהינתן אקורד מסוים, לא ניתן לבצע את המיפוי בכיוון ההפוך ולשייכו באופן חד-חד ערכי לסולם ספציפי, כי אם למספר סולמות שונים.