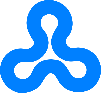
**Soloist מסמך אפיון וניתוח**

**גרסה 1.0**

🎼

**🎜 🎶 🎝**

**חנן וולט**

**‏09/10/2020**

**האוניברסיטה הפתוחה**

**תוכן עניינים**

[1 כללי 4](#_Toc53172985)

[1.1 עקרונות הפיתוח 4](#_Toc53172986)

[2 הנחות עבודה 5](#_Toc53172987)

[3 מוסכמות שמות במערכת 6](#_Toc53172988)

[1.3 מוסכמות שמות הטבלאות בבסיס הנתונים 6](#_Toc53172989)

[2.3 מוסכמות שמות רכיבי הפיתוח 6](#_Toc53172990)

[4 ארכיטקטורת המערכת 8](#_Toc53172991)

[1.4 סקירה כללית 8](#_Toc53172992)

[2.4 שכבות המערכת 9](#_Toc53172993)

[5 שכבת הלוגיקה העסקית (BLL) 10](#_Toc53172994)

[1.5 ייצוג ישויות מוסיקליות (Music Theory) 11](#_Toc53172995)

[1.1.5 משך שהייה (Duration) 12](#_Toc53172996)

[2.1.5 תווים (Notes) 15](#_Toc53172997)

[3.1.5 אקורדים (Chords) 20](#_Toc53172998)

[4.1.5 תיבות (Bars) 22](#_Toc53172999)

[5.1.5 מרווחים (Intervals) 24](#_Toc53173000)

[6.1.5 מחלקת מפעל לישויות מוסיקליות (MusicTheoryFactory) 25](#_Toc53173001)

[7.1.5 הרחבות ושירותים נוספים 26](#_Toc53173002)

[2.5 תת-שכבת שירותי מידי (MIDI) 28](#_Toc53173003)

[1.2.5 רקע – פרוטוקול MIDI וקובצי SMF 28](#_Toc53173004)

[2.2.5 ארכיטקטורת תת-שכבת שירותי ה-MIDI 29](#_Toc53173005)

[3.2.5 אוסף מספרי רצועות (MelodyTrackIndex Enumeration) 30](#_Toc53173006)

[4.2.5 אוסף כלי נגינה (MusicalInstrument Enumeration) 31](#_Toc53173007)

[5.2.5 מנשק רצועת מידי (IMidiTrack Interface) 31](#_Toc53173008)

[6.2.5 מנשק קובץ מידי (IMidiFile Interface) 32](#_Toc53173009)

[7.2.5 מחלקת מעטפת לקובץ מידי (DryWetMidiAdapter Class) 34](#_Toc53173010)

[8.2.5 מחלקת מעטפת לרצועת מידי (DryWetMidiTrackAdapter Class) 35](#_Toc53173011)

[5.2.9 מחלקת מפעל לישויות מידי (MidiFactory Class) 36](#_Toc53173012)

[3.5 הלחנת מנגינות (Composition Service) 37](#_Toc53173013)

[5.3.1 מחלקת קונטקסט ההלחנה (CompositionContext Class) 38](#_Toc53173014)

[5.3.2 מחלקת מלחין אבסטרקטי (Composer Abstract Class) 43](#_Toc53173015)

[5.3.3 הלחנה עם אלגוריתם גנטי 48](#_Toc53173016)

[5.3.4 הלחנה עם אלגוריתמים נוספים 61](#_Toc53173017)

[5.3.5 מפעל ליצירת מלחינים 62](#_Toc53173018)

[5.3.6 אינטראקציה בין הישויות בתהליך הלחנה 63](#_Toc53173019)

[5.3.7 הרחבות (ComposerExtensions) 65](#_Toc53173020)

[5.3.8 אוספי קבועים (Enums) 65](#_Toc53173021)

[6 שכבת הגישה לנתונים (DAL) 68](#_Toc53173022)

[1.6 מסד הנתונים (Database) 70](#_Toc53173023)

[1.1.6 דיאגרמת טבלאות 70](#_Toc53173024)

[2.1.6 פירוט הטבלאות 70](#_Toc53173025)

[2.6 מודלים (Domain Models) 72](#_Toc53173026)

[1.2.6 מודל משתמש (ApplicationUser Model) 72](#_Toc53173027)

[2.2.6 מודל שיר (Song Domain Model) 73](#_Toc53173028)

[3.6 מאגרים (Repositories) 74](#_Toc53173029)

[1.3.6 מנשק מאגר (IRepository Interface) 75](#_Toc53173030)

[2.3.6 מנשק מאגר שירים (ISongRepository Interface) 76](#_Toc53173031)

[3.3.6 מחלקת-אב גנרית למימוש מנשק המאגר הגנרי (EFRepository Class) 77](#_Toc53173032)

[4.3.6 מחלקת מאגר שירים (SongRepository Class) 78](#_Toc53173033)

[4.6 ניהול יחידות עבודה (Unit Of Work) 80](#_Toc53173034)

[1.4.6 מנשק יחידת עבודה (IUnitOfWork Interface) 81](#_Toc53173035)

[2.4.6 מחלקת יחידת עבודה (EFUnitOfWork Class) 82](#_Toc53173036)

[5.6 מיפוי בין אובייקטים לרשומות (EntityFramework ORM) 83](#_Toc53173037)

[1.5.6 מחלקת קונטקסט מסד הנתונים (ApplicationDbContext Class) 84](#_Toc53173038)

[2.5.6 מחלקת מיפוי ישות שיר לטבלה (SongConfiguration Class) 85](#_Toc53173039)

[6.6 מיגרציות (Migrations) 85](#_Toc53173040)

[7 שכבת התצוגה (PL) 86](#_Toc53173041)

[1.7 תצורת התוכן האפליקטיבי על השרת (App\_Data) 87](#_Toc53173042)

[1.1.7 לוגים (Logs) 87](#_Toc53173043)

[2.1.7 שירים (Songs) 87](#_Toc53173044)

[3.1.7 מנגינות (Outputs) 87](#_Toc53173045)

[4.1.7 סמינר (Seminar) 88](#_Toc53173046)

[2.7 קונפיגורציות אתחול אפליקציה (App\_Start) 88](#_Toc53173047)

[1.2.7 הגדרת חבילות (BundleConfig) 88](#_Toc53173048)

[2.2.7 רישום מחלקות פילטר (FilterConfig) 88](#_Toc53173049)

[3.2.7 הגדרת שירותי זהויות (IdentityConfig) 88](#_Toc53173050)

[4.2.7 הגדרת שירותי הרשאה (Startup.Auth) 89](#_Toc53173051)

[3.7 בקרים (Controllers) 90](#_Toc53173052)

[1.3.7 בקר דף הבית (HomeController) 90](#_Toc53173053)

[2.3.7 בקר שירים (SongsController) 91](#_Toc53173054)

[3.3.7 בקר הלחנה (CompositionController) 94](#_Toc53173055)

[4.3.7 בקר ניהול (AdminController) 95](#_Toc53173056)

[5.3.7 בקרי ניהול חשבונות משתמשים 95](#_Toc53173057)

[4.7 פילטרים (Filters) 96](#_Toc53173058)

[1.4.7 פילטר טיפול בחריגות (CustomExceptionHandler) 97](#_Toc53173059)

[2.4.7 פילטר טיפול בבקשות (ActionFilters) 98](#_Toc53173060)

[5.7 מחלקות ולידציה (Validations) 99](#_Toc53173061)

[1.5.7 בדיקת תקינות קבצים (FileUploadValidation) 99](#_Toc53173062)

[2.5.7 בדיקת תקינות טווח צלילים (PitchRangeValidation) 101](#_Toc53173063)

[6.7 מודלים לטפסים (ViewModels) 102](#_Toc53173064)

[1.6.7 מודל-תצוגה עבור שיר (SongViewModel Class) 102](#_Toc53173065)

[2.6.7 מודל-תצוגה עבור הלחנת מנגינה (CompositionModel Class) 103](#_Toc53173066)

[3.6.7 מודל-תצוגה עבור ניהול חשבונות משתמשים 103](#_Toc53173067)

[7.7 מסכים וטפסים (Views) 104](#_Toc53173068)

[8.7 אינטראקציה בין רכיבי ה-MVC 105](#_Toc53173069)

[8 דפוסי עיצוב 106](#_Toc53173070)

[1.8 דפוסי יצירה (Creational Patterns) 106](#_Toc53173071)

[1.1.8 דפוס מפעל (Factory Pattern) 106](#_Toc53173072)

[2.8 דפוסים מבניים (Structural Patterns) 108](#_Toc53173073)

[1.2.8 דפוס מעטפת מתאמת (Adapter Pattern) 108](#_Toc53173074)

[8.3 דפוסים התנהגותיים (Behavioral Patterns) 109](#_Toc53173075)

[1.3.8 דפוס אסטרטגיה אלגוריתמית (Strategy Pattern) 109](#_Toc53173076)

[2.3.8 דפוס תבנית אלגוריתמית (TemplateMethod Pattern) 110](#_Toc53173077)

[4.8 תבניות ודפוסים ארכיטקטוניים 111](#_Toc53173078)

[1.4.8 מודל-תצוגה-בקרה ((MVC 111](#_Toc53173079)

[2.4.8 מיפוי אובייקטי-רלציוני (ORM) 111](#_Toc53173080)

[3.4.8 הפשטת אינטראקציה מול DB (Repository & Unit Of Work) 112](#_Toc53173081)

[9 תיאור שינויים עתידיים 113](#_Toc53173082)

[1.9 שינויים תשתיתיים 113](#_Toc53173083)

[1.1.9 שינויים תשתיתיים הקשורים ל-DB 113](#_Toc53173084)

[2.1.9 שינויים בפלטפורמה 114](#_Toc53173085)

[2.9 שינויים אפליקטיביים/פונקציונאליים 114](#_Toc53173086)

[1.2.9 הוספת אלגוריתם הלחנה 114](#_Toc53173087)

[2.2.9 עדכון האלגוריתם הגנטי הקיים 115](#_Toc53173088)

[10 נספח: רקע תאורטי של המוסיקה המערבית 116](#_Toc53173089)

# כללי

מסמך זה מכיל את פרטי מימוש העיצוב והתיכון של מערכת Soloist כפי שאופיינה 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 כאפליקציית Web באמצעות ASP.NET MVC 5 עבור ניהול צד השרת, ו-HTML, CSS ו-Javascript עבור ניהול צד הלקוח, ללא שימוש בספריות frontend ייעודיות, למעט השלמה של פונקציונאליות בסיסית עם Bootstrap ו-JQuery.

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

כמו כן, המימוש כולל שימוש גם בספריות חיצוניות המספקות שירותים חיוניים מסוימים למערכת כגון ספריית DryWetMIDI עבור ניהול וכימוס אירועי MIDI, ספריית Autofac עבור ניהול תלויות (Dependency Injection & IoC), וספריית SendGrid עבור שליחת מיילים.

## עקרונות הפיתוח

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

1. **ממשק 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 לעטוף ולהסתיר את כל הידע התאורטי המוסיקלי וכל הלוגיקה האלגוריתמית של הקומפוזיציה מאחורי ממשק מופשט – "קופסא שחורה", כך שמשתמשים יוכלו לחבר מנגינות חדשות על-פי טעמם בכמה לחיצות כפתור.
2. **גמישות לשינויים** – ארכיטקטורת המערכת צריכה לאפשר שינויים ושיפורים עתידיים ותחזוקה שוטפת בקלות יחסית ללא עדכונים מהותיים. לטובת עיקרון זה, בכל חלקי הפיתוח מוטמעים דפוסי עיצוב (Design Patterns) ועקרונות ה-[SOLID](https://en.wikipedia.org/wiki/SOLID) עבור תכנות מונחה עצמים.
3. **אי-תלות בפלטפורמה**  – המערכת צריכה להיות חופשיה מתלות ברכיבי החומרה, מערכת ההפעלה וסוג היישום (יישום שולחני / Web / אפליקציית Smartphone). לטובת אי-תלות זו המערכת עוצבה בחלוקה לשכבות – שכבת תצוגה/יישום, שכבת הלוגיקה העסקית, ושכבת ניהול הגישה לנתונים. בשלב ראשוני הוחלט לפתח שכבת תצוגה כיישום Web, כך שיתאפשר שימוש במערכת מכל מחשב נייח/נייד/סמארטפונים ללא תלות במערכת ההפעלה וברכיב החומרה. אם בעתיד יידרש ממשק משתמש ייעודי חדש/נפרד למחשב שולחני / סמארטפון / טאבלט וכד', ניתן יהיה לעשות שימוש בתשתית הקיימת של שכבות ניהול הגישה לנתונים והלוגיקה העסקית ורק לספק מימוש חדש לשכבת התצוגה/יישום.

# הנחות עבודה

הפיתוח נעשה תחת הנחות העבודה הבאות –

**חומרה**

* המערכת מיועדת להפעלה בעיקר ממחשבי PC. אמנם היישום הוא Web-י, כך שיש תמיכה בכל המכשירים ללא תלות במערכת ההפעלה ובסוג המכשיר (PC, Tablet, Smartphone,...) אולם הפיתוח מוכוון למחשבי PC, אז לקבלת תצוגה מיטבית מומלץ לעשות שימוש במערכת מתוך מחשבים נייחים/ניידים ולא מתוך טלפונים חכמים.

**הנחות על קובצי ה-MIDI**

הפורמט המרכזי לקובצי השירים הוא קובצי MIDI. דיוק מנומק לבחירה בפורמט זה מפורט במסמך ה[אפיון והניתוח](https://github.com/cwelt/Soloist/blob/master/Design/OOA-Object-Oriented-Analysis.pdf) ובמסמך זה בסעיף [רקע – פרוטוקול MIDI וקובצי SMF](#_רקע_–_פרוטוקול). להלן הנחות נוספות בדבר תוכן הקבצים –

* קבצי MIDI בפורמט 0, המרכזים את תפקידי כל כלי הנגינה ביחד עם ה-Metadata של נתוני כותרת הקובץ ברצועה יחידה, אינם נתמכים ע"י המערכת. המערכת תומכת רק בפורמטים 1 ו-2, המבצעים הפרדה בין הרצועות השונות וכן בין רצועות הנגינה לבין רצועת הכותרת. פירוט נוסף על פורמטים בקובץ MIDI ניתן למצוא בקישור הבא –

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

* המערכת תומכת ב- 16 רצועות לכל היותר (לא כולל את רצועת הכותרת). במידה ויש צורך לטפל בקובץ MIDI הכולל בנוסף לרצועת הכותרת 17 רצועות או יותר, יש לעדכן תחילה את הקובץ כך שירכז את אירועי ה-MIDI ב-16 רצועות נגינה לכל היותר.
* עבור כלי הנגינה שיוגדר לנגן מנגינות חדשות שייווצרו ע"י המערכת, יש תמיכה בכלי נגינה מסט הערכים של ה-General Midi בטווח 0 עד 112 (עד משפחת הכלים האתניים כולל). טווח זה כולל את כל כלי הנגינה הסטנדרטיים, ללא אפקטים קוליים סינטטיים וללא כלי הקשה. פירוט נוסף על ערכי ה-General Midi ניתן למצוא בקישור הבא –

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

**הנחות על מגבלות ייצוג / זיכרון**

המערכת יוצאת מנקודת הנחה שהשירים המועלים למערכת הם בעלי הגדרות קצב ומקצב "הגיוניים", במובן שהם ניתנים לנגינה ע"י בן אנוש. תחת הנחות סבירות אלו, מוגדרות המגבלות הבאות –

* ערך ה-BPM – Beats Per Minute שמגדיר את מספר הפעימות בדקה של השיר, וכנגזרת מכך קובע את מהירות הקצב, מיוצג ע"י byte, משמע שטווח הערכים שלו נע בין 0 ל-255.
* משכי שהייה (כגון רבע, שמינית וכו') המיוצגים במערכת ע"י שני משתנים נפרדים המרכיבים את המונה והמכנה, מיוצג גם הוא במונה ובמכנה ע"י משתני byte וטווחי ערכים אלו בהתאם.

# מוסכמות שמות במערכת

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

## מוסכמות שמות הטבלאות בבסיס הנתונים

שמות הטבלאות והשדות יהיו שתיהם ב-Pascal Case (אות רישא גדולה בתחילת כל מילה). שמות טבלאות יהיו ברבים ויהיו בעלי שמות תואמים עד כמה שניתן למחלקות התואמות להן ברכיבי הפיתוח, במידה וישנן כאלה. למשל עבור מחלקת Song המייצגת שיר, שם הטבלה יהיה Song**s**.

## מוסכמות שמות רכיבי הפיתוח

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

1. [C# Coding Standards and Naming Conventions](https://bit.ly/3gSoOGk)
2. <https://docs.microsoft.com/en-us/dotnet/standard/design-guidelines/naming-guidelines>

בסעיף 'מוסכמות כתיבה' שלהלן נביא את עיקרי הדברים והבחירות שבוצעו בפועל בפיתוח המערכת מבין האפשרויות המומלצות בקווי ההנחיה הסטנדרטיים, כפי שמתוארים בקישורים שלעיל.

**הערות מקדימות**

1. ביאור: בתיאור מוסכמות 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, ו - Camel Case היא צורת כתיבה שבה שם העצם/פועל/מילה הראשונים נכתבים דווקא עם אות קטנה, ואילו המילים העוקבות פותחות עם אות רישית (אות גדולה), (למשל: chordProgressionFilePath). בכדי שיהיה זה ברור מידית בהקשר הרלוונטי, בהמשך פשוט נרשום את המושגים שלעיל בצורת הכתיבה שהם מייצגים כך: PascalCase ו-camelCase.
2. התיאור ברשימת מוסכמות הכתיבה המפורטות בהמשך הינו תיאור של מוסכמות, ולא הנחיות מחייבות – אין כאן נכון ולא נכון, אלא פשוט קווי הנחייה שכדאי להיצמד אליהם בכדי שהקוד יהיה אחיד וברור ישירות מתוך המוסכמות. ואכן, ישנם מקרים חריגים יוצאים מן הכלל שבהם לא מיושמים המוסכמות שלעיל. למשל –

* משתנים פרטיים קבועים (private const…) – שהם גם פרטיים וגם קבועים, אז מצד אחד ע"פ המוסכמות הם אמורים להירשם בפורמט משתנים פרטיים עם קו-תחתון תחילי ו- camelCase(\_somePrivateFieldName), ומצד שני הם קבועים ולכן אמורים להירשם ללא קו-תחתון תחילי וב-PascalCase. במקרים סתורים שכאלה, אין כלל קבוע, לעיתים השדה יירשם במוסכמה הראשונה (אם למשל נורא חשוב להדגיש שמדובר בשדה פרטי שאינו אמור להיות מוכר כלל לרכיבי פיתוח חיצוניים) ולעיתים השנייה (אם למשל העובדה שמדובר בקבוע חשובה יותר להדגשה מהעובדה שמדובר במשתנה שהוא פרטי לגוף המחלקה).
* בפלטפורמת ASP.NET MVC לפיתוח יישומי ה-Web, נהוגה דווקא מוסכמת כתיבה שונה עבור משתנים פרטיים שמקבלים שם דווקא שמות עם camelCase ללא קו-תחתון תחילי, למשל: private DbContext db (ולא private DbContext \_db), על-כן במרחב השמות של פיתוח המנשק ה-Web-י למשל, ייתכנו חריגות קלות שכאלה ממוסכמות הכתיבה שלעיל.

מעבר למקרים החריגים שיוצאים מן-הכלל פה ושם כמו שתי הדוגמאות שלעיל, קור-המקור של המערכת נצמד באופן עקבי למוסכמות הכתיבה המפורטות מיד בעמוד הבא.

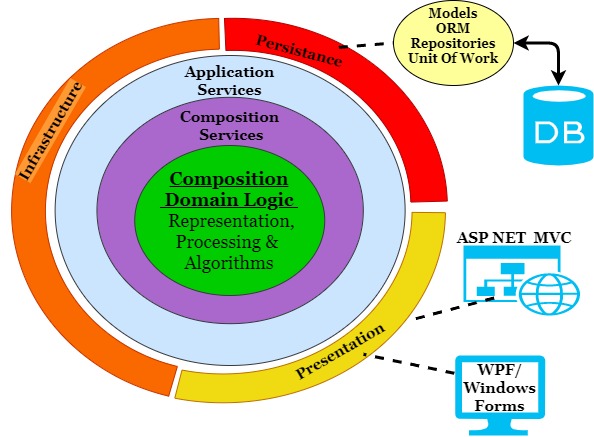
**רשימת מוסכמות הכתיבה של שמות רכיבי הפיתוח** –

* **מרחבי שמות** **(Namespaces)** יתחילו עם הקידומת CW.Soloist<ProjectName>: CW עבור רשי התיבות של שם המחבר[[1]](#footnote-1), Soloist עבור שם המערכת, ולאחר מכן שם הפרויקט (שם יחידת ה-Assembly) המרכזי של רכיבי הפיתוח השייכים למרחב השמות הזה. למשל עבור שכבת הגישה לנתונים המרוכזת בפרויקט DataAccess, כלל מרחבי השמות בפרויקט יהיו תחת עץ המשפחה של מרחב השמות: CW.Soloist.DataAccess. מוסכמה זו חלה גם על ההיררכית הפנימית, למשל, הלוגיקה לניהול Repositories בשכבת הנתונים יכולה להיות מרוכזת תחת מרחב השמות CW.Soloist.DataAccess.Repositories. כך ברור מידית מההקשר מהו מקור הספרייה/רכיב פיתוח (ספרייה חיצונית/ספרייה השייכת למערכת) וכן במקרה שאכן מרחב השמות שייך למערכת, ניתן לגזור מיד מהשם מהי השכבה הרלוונטית, ולאיזה ענף בהיררכיה הפנימית של אותה שכבה רכיב הפיתוח שייך.
* **רכיבי פיתוח עם שמות בצורת PascalCase**: מרחבי שמות (Namespaces), מחלקות, מנשקים (Interfaces), מתודות (שיטות), בנאים (Constructors), מאפיינים (Properties), משתני הפנייה למתודות (Delegates), משתנים קבועים וכן טיפוסי קבוצות בנות-מנייה של קבועים (Enumerations) – כל אלו נכתבים בשיטת PascalCase.
* **רכיבי פיתוח עם שמות בצורת camelCase**: פרמטרים למתודות, משתנים מקומיים, ושדות (משתני מופע שאינם נמנים בין ה-Properties של המחלקה/מנשק) – כל אלו נכתבים בשיטת camelCase.
* **שמות משמעותיים**:להוציא אינדקסים שכל תפקידם הוא למנות איטרציות בלולאות, כל יתר רכיבי הפיתוח – משתנים, שדות, מתודות וכו' יישאו שם משמעותי המבהיר היטב את טיב הרכיב.
* **משתנים פרטיים** ייכתבו עם קו תחתון תחילי '\_' (underscore character) לפני שם המשתנה למשל עבור משתנה פרטי compositor:private Compositor \_compositor;
* **מנשקים** (Interfaces) ייכתבו עם תחילית של האות הרישית I בכדי להדגיש שמדובר ב-Interface. לדוגמה: אם נרצה להגדיר פונקציונאליות של הלחנה (קומפוזיציה) תחת ישות Compositor שתהיה אחראית על פונקציונאליות זו, ונבחר לרכז את ההגדרה המופשטת של פונקציונאליות זו במנשק ייעודי, שם הולם למנשק שכזה יכול להיות למשל ICompositor.
* **מתודות** יקבלו שמות משמעותיים שמייצגים את פעולתן בגוף שני. שם הולם למשל, למתודה שאחראית על הלחנה הוא Compose. שמות טובים יבהירו מידית את מהות אקט המתודה. למשל המתודה ChangePitchForARandomNote אחראית על שינוי גובה צליל לתו אקראי.
* **משתנים בוליאניים ומתודות בוליאניות** ייכתבו עם שם המייצג את מהות הפסוק הלוגי / אמרה לוגית שערך האמת שלהם הוא זה שמוחזר ע"י 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. פונקציונאליות זו משתמעת ישירות מהמוסכמה.
* **רכיבי פיתוח מרכזיים המשתתפים במימוש דפוסי עיצוב** יכילו בשמם את תפקידם בקונטקסט מימוש דפוס העיצוב. למשל – מחלקות שאחראיות על יצירת אובייקטים אחרים יכילו בשמם את המושג Factory, ומחלקה האחראית לעטוף איזשהו מנשק, להסתירו ולספק במקומו מנשק אחר, יכילו בשמן את המושג Adapter.

# ארכיטקטורת המערכת

## סקירה כללית

לטובת הסדר הטוב במערכת, המודולריות שלה, אמינותה והגמישות שלה לתחזוקה שוטפת ושדרוגים עתידיים, המערכת נבנתה בחלוקה לשכבות לוגיות ופיזיות שונות בהשראת ארכיטקטורת ה-3-Tier Architecture, אולם להבדיל מהגישה הקלאסית של ארכיטקטורת שלוש השכבות – שכבת תצוגה, שכבת אפליקציה ושכבת גישה לנתונים שבה בד"כ שכבת הגישה לנתונים היא השכבה המרכזית שעיצוב המערכת נעשה סביבה (Data-Driven Design), במערכת Soloist הנתונים והגישה אליהם הם דווקא רכיב שולי בקונטקסט של האפליקציה: הרכיב המרכזי שנמצא בלב מערכת Soloist הוא דווקא ה-Domain – הלוגיקה לייצוג צלילים ומנגינות, העיבוד שלהם וכל האלגוריתמים המתוחכמים שיחד מרכיבים את השירות המרכזי שהמערכת מספקת – הלחנה אוטומטית של מנגינות. לפיכך עיצוב כל המערכת נעשה סביב תחום זה של הלחנת המנגינות בעיצוב שמונחה תחת תחום היישום (Domain-Driven Design). באופן סכמתי, נח לתאר את הארכיטקטורה של המערכת שנבנתה בעיצוב זה כסדרה של טבעות, כאשר כל טבעת מייצגת שכבה, והשכבות שמיוצגות ע"י הטבעות הפנימיות מספקות שירותים לשכבות שמיוצגות ע"י הטבעות החיצוניות יותר –



נסקור כעת בקצרה את השכבות המרכזיות המרכיבות את המערכת כפי שהן מתוארות בתרשים שלעיל, ובפרקים הבאים נעבור על כל אחת מהשכבות בנפרד ועבור כל שכבה נתאר באופן מפורט את כל רכיבי הפיתוח שמרכיבים אותה – מחלקות, מנשקים, אינטראקציות וכו'.

## שכבות המערכת

להלן תקציר של השכבות הלוגיות והפיזיות השונות המרכיבות את המערכת. זוהי סקירה מתומצתת בלבד. בפרקים הבאים כל אחת מהשכבות שלעיל מתוארת בנפרד בפרטי פרטים.

1. **שכבת הלוגיקה העסקית של האפליקציה (Domain Business Logic Layer)** – שכבה זו מהווה את הלב והמח של המערכת. היא מכילה את כל הלוגיקה העסקית והאלגוריתמים הדרושים לייצוג, עיבוד והלחנה של מנגינות. בכדי לבודד את כל הלוגיקה המורכבת הזו ליחידה אטומית עצמאית שאינה תלויה ביתר רכיבי הפיתוח, שכבה זו נבנתה בגישה מוכוונת-שירות (service oriented): הוגדרו מנשקים חיצוניים פשוטים (high-level) של כל השירותים ששכבה זו אחראית לספק, וכל הלוגיקה למימוש מנשקים אלו כמוסים ומוסתרים מיישומים המשתמשים בהם. בתרשים, המנשק שחושף את השירותים ליתר השכבות מיוצג ע"י שטח הטבעת הסגולה, ואילו המימוש של כל השירותים הללו מיוצג ע"י השטח הירוק שבמרכז התרשים, ואכן, לב המערכת הא מימוש הלוגיקה של כל השירותים הללו.
2. **שכבת הגישה לנתונים** (**Data Access Layer) –** שכבה זו אחראית על ניהול הגישה לנתונים על כל המשתמע מכך – ריכוז מחלקות ה-Domain Models, אינטראקציה מול מסד הנתונים, מיפוי בין הטבלאות והרשומות שבבסיס הנתונים לבין הישויות והאובייקטים המיוצגים במחלקות ה-Domain Model, ואספקת שכבת הפשטה מעל הגישה לנתונים באמצעות Repositories ומנהל ישויות המתפקד כ-Unit Of Work, מה שמבטיח אי-תלות של השכבות החיצוניות בבסיס הנתונים, ואף אי-תלות בטכנולוגיית ה-ORM המממשת את המיפוי שבין הטבלאות למחלקות. כל הגישות לבסיס הנתונים מבוצעות אך רק באמצעות השכבה הזו. אף שכבה אחרת אינה ניגשת לבסיס הנתונים ישירות. ארכיטקטורה זו מבטיחה שהחלפת מוצר ה-ORM או בסיס הנתונים יכולה להיעשות באופן "שקוף" לשכבות האחרות, כל עוד שכבת הגישה לנתונים דואגת לשמר את מימוש המנשק 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 גראפי (GUI), , האזנה לקלט המשתמש ומתן מענה לבקשות שלו תוך אינטראקציה עם שכבות הגישה לנתונים לאחזור ועדכון רשומות וכן שכבת הלוגיקה העסקית של האפליקציה לטובת צריכת השירותים שהמשתמש מבקש. בידוד התצוגה מהלוגיקה העסקית ומהגישה לנתונים מאפשר להגדיר תצוגות שונות ואף אפליקציות שונות (מנקודת המבט של המשתמש) המתבססות על השירותים שכבות הלוגיקה והנתונים ומציעות חוויות משתמש שונות. במימוש הנוכחי הוחלט לממש את שכבת התצוגה בממשק Web-י בפלטפורמת ASP.NET MVC, אולם הוגדרו גם פרויקטים נוספים עם תצוגה אלטרנטיבית – האחד פרויקט Windows Forms שנבנה כאב-טיפוס למערכת, ומציג ממשק חלונאי ביישום שולחני (Desktop Application) מתאים למחשבי PC של Windows. השני הוא פרויקט Console Application, החושף ממשק משתמש מינימאלי של קונסול עם מפרש לשורות פקודה. שני פרויקטים אלו מתבססים על שירותי שכבת הלוגיקה העסקת וממחישים אלטרנטיביות שונות למימוש שכבת תצוגה.

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

|  |  |  |  |
| --- | --- | --- | --- |
| # | שכבה | פרויקט | מרחב שמות (Namespace) |
| [1](#_שכבת_הלוגיקה_העסקית) | לוגיקה עסקית (BLL) | CompositionService | CW.Soloist.CompositionService |
| [2](#_שכבת_הגישה_לנתונים) | גישה לנתונים (DAL) | DataAccess | CW.Soloist.DataAccess |
| [3](#_שכבת_התצוגה_(PL)) | תצוגה (PL) | WebApplication | CW.Soloist.WebApplication |

# שכבת הלוגיקה העסקית (BLL)

שכבה זו מהווה את הלב והמח של המערכת, וכוללת את כל הלוגיקה העסקית הנדרשת לייצוג ועיבוד של תווים, ואוסף אלגוריתמים להלחנת מנגינות חדשות. שכבה זו מוגדרת בשלמותה כספרייה בפרויקט ייעודי המהווה יחידת אסמבלי עצמאית: CompositionService. בהתאם (וע"פ [מוסכמות השמות במערכת למרחבי שמות](#NamespaceConventions)), כלל מרחבי השמות (namespaces) השייכים לשכבה זו מושרשים תחת מרחב השמות: CW.Soloist.CompositionService.

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

1. **ייצוג ישויות מוסיקליות** – ייצוג של תווים, מרווחים, אקורדים, סולמות, תיבות וכו'.
2. **מימוש לוגיקה אלגוריתמית להלחנת מנגינות** – כתיבת אלגוריתמים לעיבוד וניתוח של מנגינות, יצירה ועדכון של המנגינות, הערכת המנגינות הנוצרות ועוד, תוך שימוש בישויות המוסיקליות כפי שמיוצגות במערכת.
3. **ניהול אינטראקציה ומניפולציה של קבצי MIDI –** קריאת קבצי MIDI, פענוח תוכן קבצי MIDI והמרת אירועי ה-MIDI המפוענחים לישויות המוסיקליות כפי שמיוצגות במערכת, יצירת קבצי MIDI חדשים, עדכון רצועות בקבצי MIDI קיימים, ניגון קבצי ה-MIDI כקובצי שמע, ועוד.
4. **עטיפת הלוגיקה בשירות פשוט –** כימוס הלוגיקה המורכבת המוגדרת בשכבה זו ואספקתה כשירות במנשק פשוט וקל לשימוש (high-level) שיהיה זמין לשכבות חיצוניות ויסתיר את הלוגיקה הפנימית.

בכדי 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 שלעיל. להלן סקירה קצרה של הרכיבים השונים המרכיבים את שכבת הלוגיקה העסקית עם מרחב-השמות הפנימי של כל רכיב (תת-שכבה/מודול) –

|  |  |  |
| --- | --- | --- |
| # | תת-הרכיב בשכבת הלוגיקה העסקית | שם מרחב שמות (Namespace) של הרכיב |
| 1 | ניהול האינטראקציה בין הרכיבים השונים של השכבה ועטיפתם במנשק פשוט שיהיה זמין כשירות לשכבות החיצונית | CW.Soloist.CompositionService |
| 2 | ייצוג ישויות מהתאוריה המוסיקלית | CW.Soloist.CompositionService.MusicTheory |
| 3 | מימוש אלגוריתמים להלחנת מנגינות | CW.Soloist.CompositionService.Compositors |
| 4 | אינטראקציה עם קבצי MIDI | CW.Soloist.CompositionService.Midi |
| 5 | אוספי עזר של קבועים | CW.Soloist.CompositionService.UtilEnums |

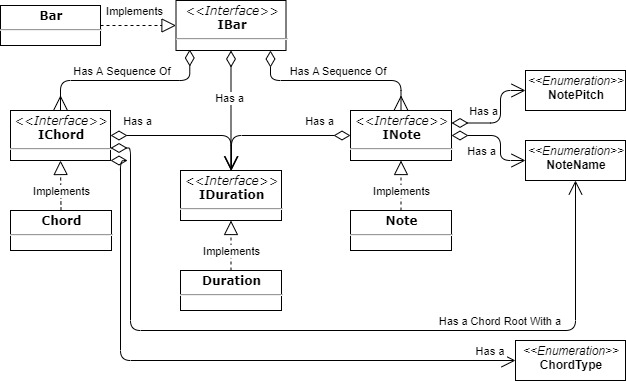
בעמודים הבאים מתוארים הרכיבים השונים בפירוט.

## ייצוג ישויות מוסיקליות (Music Theory)

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

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

|  |  |  |  |
| --- | --- | --- | --- |
| # | תפקיד / ישות מוסיקלית | סוג רכיב פיתוח | שם הרכיב |
| 1 | ייצוג משך שהייה | Interface | [IDuration](#_IDuration_Interface) |
| 2 | Class | [Duration](#_Duration_Class) |
| 3 | ייצוג תווים | Enumeration | [NotePitch](#_Enumeration_NotePitch) |
| 4 | Enumeration | [NoteName](#_NoteName_Enumeration) |
| 5 | Interface | [INote](#_INote_Interface) |
| 6 | Class | [Note](#_Note_Class) |
| 7 | ייצוג אקורדים | Enumeration | [ChordType](#_ChordType_Enumeration) |
| 8 | Interface | [IChord](#_IChord_Interface) |
| 9 | Class | [Chord](#_Chord_Class) |
| 10 | ייצוג תיבות | Interface | [IBar](#_IBar_Interface) |
| 11 | Class | [Bar](#_Bar_Class) |
| 12 | ייצוג מרווחים | Enumeration | [PitchInterval](#_PitchInterval_Enumeration) |
| 13 | יצירת ישויות מוסיקליות | Class | [MusicTheoryFactory](#_מחלקת_מפעל_לישויות) |
| 14 | הרחבות ושירותים כלליים עבור הישויות השונות | Class | [MusicTheoryExtensions](#_MusicTheoryExtensions) |
| 15 | Class | [MusicTheoryServices](#_MusicTheoryServices) |

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

**הערה לחסרי רקע תאורטי מוסיקלי**: מודול זה אחראי כאמור על ייצוג של ישויות מוסיקליות. בתיעוד מוזכרים לא פעם מונחים מוסיקליים כגון תווים, סולמות, וכד'. אמנם אין הכרח ברקע מוסיקלי קדים בכדי לקבל את ה-"Big Picture" מבחינת ה-Design הכללי, אולם כדי להפיק ממסמך זה את המרב, מומלץ שתהיה הכרות לפחות עם המושגים הבסיסיים. למטרה זו, בסוף מסמך זה מצורף [נספח: רקע תאורטי של המוסיקה המערבית](#_נספח:_רקע_תאורטי_1) בן 4 עמודים בלבד המיועד לחסרי רקע מוסיקלי ומספק סיכום תמציתי של הידע המוסיקלי הנדרש למסמך זה. על-כן, לחסרי רקע מוסיקלי, מומלץ לעבור על 4 עמודים אלו טרם קריאת המשך הפרק, או לחלופין, אם תוך כדי מעבר על פרטי המימוש נתקלים במושג לא ברור, כדאי פשוט לעבור לנספח שבסוף המסמך באופן נקודתי בכדי לקבל הבהרה על המושג להשלמת התמונה המלאה.

### משך שהייה (Duration)

**רקע**: בהינתן איזושהי מנגינה המורכבת מרצף צלילים, כל צליל מתנגן לאיזשהו פרק זמן סופי. בתאוריה המוסיקלית המערבית, במקום לייצג ולאמוד פרק זמן זה ביחידות זמן אבסולוטיות (כגון שניות או מילישניות), מייצגים את משכי השהייה של הצלילים השונים באופן רלטיבי ביחס למשך השהייה הכולל של תיבה שלמה אחת. אם למשל משקל התיבה מוגדר להיות ארבעה רבעים (4/4), אזי משך שהייה של חצי תיבה (1/2) משמעו משך שהייה של שני רבעים (שני פעימות עבור משקל 4/4). בייצוג זה, משך השהייה האבסולוטי של פעימה יחידה אינו רלוונטי לייצוג משך השהייה. משך השהייה האבסולוטי נקבע בהתאם ל-BPM (Beats Per Minute) – מספר הפעימות שנכנסות בדקה אחת, אולם אין זה מעניינם של הצלילים במנגינה, שמבחינתם צריכים רק לדעת מה משך השהייה שלהם ביחס לתיבה הכילה אותם. לפיכך, בכדי לייצג משך שהייה כל שנדרש הוא שבר – מספר ממשי המייצג את אותו חלק יחסי. 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 הממשי המהווה את היחס עצמו, הוחלט לממש את משך השהייה כך שיכיל גם את המונה והמכנה שמרכיבים את השבר (מנת השבר היא המספר הממשי המהווה את היחס).

#### [IDuration Interface](https://github.com/cwelt/Soloist/wiki/IDuration)

**המנשק** [IDuration](https://github.com/cwelt/Soloist/wiki/IDuration) **מייצג משך שהייה. להלן החתימות של רכיבי הפיתוח שהוא מגדיר –**

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 ב**[**דף ה-**Wiki **של המנשק ב-**Github](https://github.com/cwelt/Soloist/wiki/IDuration)**. למען הסדר טוב מובא כאן תקציר התיעוד –**

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

#### [Duration Class](https://github.com/cwelt/Soloist/wiki/Duration)

**המחלקה** [Duration](https://github.com/cwelt/Soloist/wiki/Duration) **מממשת את המנשק** [IDuration](https://github.com/cwelt/Soloist/wiki/IDuration)**. מעבר למאפיינים והמתודות במנשק, מחלקה זו מגדירה גם בנאים, מתודות נוספות והעמסות על אופרטורים של השוואה בין משכי שהייה. תיעוד מלא ומפורט של כלל הרכיבים במחלקה ניתן למצוא הן בגוף קוד המקור וכן** [**בדף ה-**Wiki **של האפליקציה ב-**.Github](https://github.com/cwelt/Soloist/wiki/Duration) **למען הסדר טוב מובא כאן תקציר התיעוד של הרכיבים הנוספים השייכים למחלקה עצמה (אלו שאינם מוגדרים במנשק** IDuration**) –**

קבועים**: הגדרת הערכים הקבועים של משכי השהייה 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 מממשת** GCD **לפישוט שבר פשוט של משך שהייה ע"י צמצום במכנה משותף מקסימאלי –**

// 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) {…}

**העמסות** **אופרטורים**: מחלקה זו מגדירה מחדש את האופרטורים הבאים להשוואה בין משכי שהייה כך שניתן להשוות ביניהם 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) {…}

בנאים: **אחד מבוסס על מונה ומכנה, ושני בנאי העתקה, שניהם עם אפשרות לצמצום עם** GCD **–**

// Constructs a duration based on a numerator and a denominator.

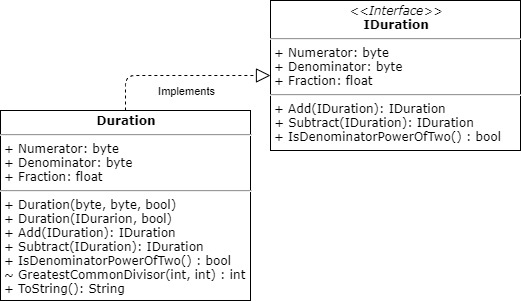
public Duration(byte numerator = 1, byte denominator = 4, bool reduceToLowestTerms = true){…}

// Constructs a duration based on an existing duration

public Duration(IDuration duration, bool reduceToLowestTerms = true) {…}

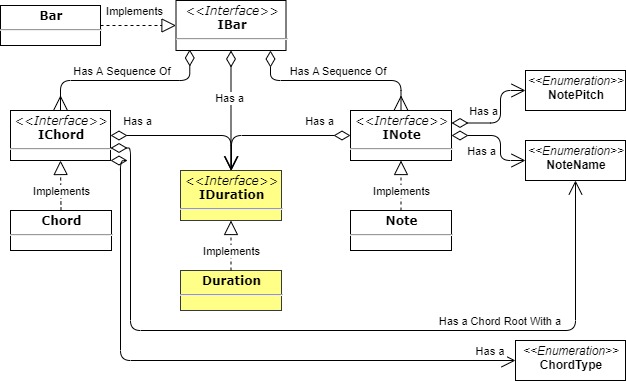
.

**דיאגרמת Class Diagram עבור המנשק והמחלקה של ייצוג משך שהייה**  –



**קשרים עם מחלקות אחרות**  –

המנשק IDuration שמחלקה זו מממשת מהווה יחידה בסיסית המשתתפת בהרכבת יישויות מוסיקליות אחרות: הן לתווים יש משך שהייה, הן לאקורדים והן לתיבות (משך השהייה של תיבה גדול או שווה מסך משכי השהייה של כל התווים (אקורדים) שהיא מכילה. להלן שוב דיאגרמת המחלקה המתארת את הקשר בין הישויות המוסיקליות המרכזיות, עם דגש על הקשר שלהן לישות משך-שהייה –

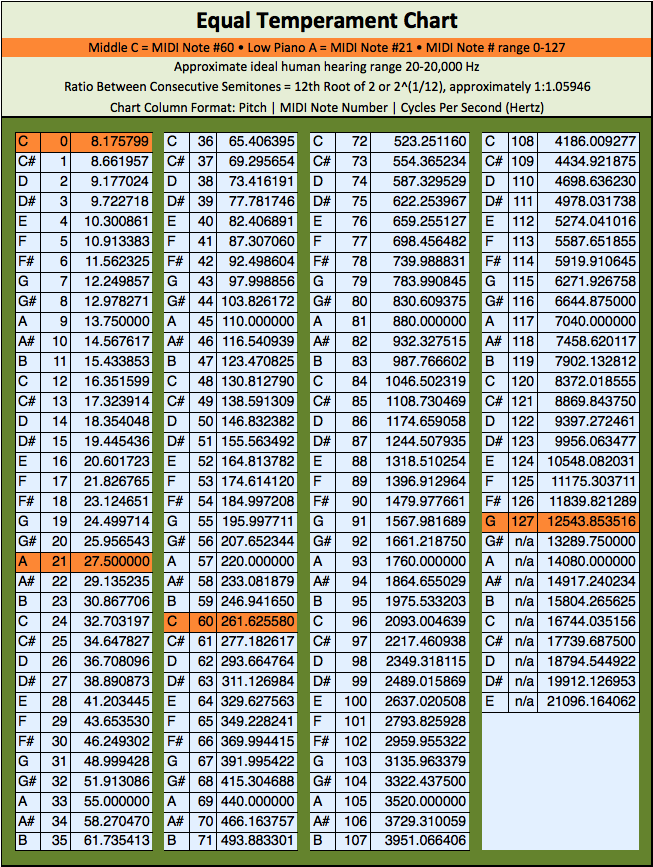


### תווים (Notes)

**רקע**: בכדי לייצג תו בודד יש צורך בשני מאפיינים: האחד – גובה הצליל, והשני – משך השהייה שלו. משך השהייה מיוצג במערכת ע"י המנשק [IDuration](https://github.com/cwelt/Soloist/wiki/IDuration) (ראה [ייצוג משך שהייה (Duration)](#_ייצוג_משך_שהייה)). לפיכך נותר להגדיר ייצוג לגובה צליל, שאותו נגדיר באמצעות Enumeration המכיל ייצוג לכל הצלילים השונים הממופים ב-MIDI. בנוסף, לטובת קריאות, בהירות וסיווג טקסטואלי לתווים, נגדיר Enumeration לייצוג שמות התווים השונים (דו, רה, מי, ...) ולבסוף נאדר אבני-בניין אלו יחד ונגדיר מנשק INote המייצג תו, עם מאפייני משך השהייה וגובה הצליל, ומחלקה Note המממשת מנשק זה.

#### [NotePitch Enumeration](https://github.com/cwelt/Soloist/wiki/NotePitch)

**הסטנדרט של** MIDI **מגדיר מיפוי בין ערכים בתחום** [0-127] **לבין 128 גבהי צלילים שונים, הכוללים בין היתר את כל 88 הצלילים המופקים ע"י מקלדת פסנתר סטנדרטית. להלן המיפוי המציג עבור כל אחד מגבהי הצלילים את שם התו, את הערך המספרי הממופה לו ב-** MIDI **ואת התדירות של הצלילים ביחידות הרץ –**



**מיפוי זה שימושי ביותר, שכן** MIDI **הוא אחד הסטנדרטיים המרכזיים לייצוג תווים וצלילים בתכניות מחשב (אם לא** "ה**סטנדרט" בהא הידיעה) ולכן ייצוג גבהי צלילים ע"פ אותם ערכים קריטי לטובת מציאת שפה אחידה ומשותפת עם ספריות חיצוניות.**

**נכון לזמן כתיבת המערכת, מאגר הספריות הסטנדרטיות של פלטפורמת ה-** .NET **לא כללה ספרייה לעבודה מול הסטנדרט של** MIDI. **עם זאת, ישנן ספריות חיצוניות רבות, ובאחת מהן אכן נעשה שימוש לטובת עיבוד ואינטראקציה מול קבצי** MIDI**, אולם לטובת ייצוג האספקט התאורטי המוסיקלי נטו (ייצוג גבהי צלילים), יהיה זה חבל להיות תלויים במימוש ספציפי כזה או אחר של אחת מספריות אלו, כשכל שנדרש הוא לייצג תווים. לפיכך התקבלה ההחלטה פשוט לממש מיפוי ישירות כאוסף קבועים, דהיינו כ-**Enumeration. **אוסף זה מוגדר בקובץ** [NotePitch.cs](https://github.com/cwelt/Soloist/wiki/NotePitch) **–**

public enum NotePitch

{

// Pitches

CMinus1 = 0,

CSharpMinus1 = 1,

...

FSharp9 = 126,

G9 = 127,

// Special notes

HoldNote = 128,

RestNote = -1

}

**מוסכמת (קונבנציית) השמות שניתנו לגובהי הצלילים הם שם התו מלווה במספר האוקטבה (ע"פ המיפוי ב-**MIDI**), למשל** G9 **משמע התו** "G" **("סול") באוקטבה התשיעית. עבור צלילים שדורשים סימני התק (הצלילים המנוגנים ע"י הקלידים השחורים במקלדת פסנתר) נבחר לקרוא לשם התו עם סימן התק של דיאז (#) –** Sharp, **אולם זהו עניין של מוסכמה בלבד. כמובן שהצליל המיוצג ע"י** FSharp **(פה דיאז) הוא בדיוק אותו הצליל המיוצג ע"י** GFlat **(סול במול), פשוט מטעמי נוחות, הוחלט להידבק לשיטה רישום אחת ונבחרה הראשונה. השם כאן ממילא 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 תווי ההיתק עם דיאזים ובין אם עם במולים.**

**בנוסף לערכי גבהי הצלילים המוגדרים ב-** MIDI**בתחום** [0-127]**, הוגדרו שני ערכים נוספים הדרושים לייצוג תווים "מיוחדים":**

1. **HoldNote** – **מייצג תו-קשת, דהיינו תו שאינו מנגן צליל חדש אלא משמר את ניגון הצליל שקדם לו ובכך למעשה מאריך את משך השהייה שלו, כאשר משך תוספת ההארכה הוא משך השהייה של תו ה-**HoldNote**.**
2. **RestNote – מייצג תו השתק של הפסקה ("שקט"). משמעות תו זה היא שלכל אורך משך השהייה שלו משתרר שקט (אף צליל אינו מתנגן).**

**לשני תווים מיוחדים אלו אין באמת גובה צליל משל עצמם, אלא רק משך שהייה. כתוצאה מכך, בשיטות הייצוג הנוכחית של תווים שנבחרה בפרויקט, שבה כל תו מורכב מגובה צליל ומשך שהייה, יוצא שערך גובה הצליל הוא מנוון וחסר משמעות. בכדי לחסוך את הצורך בהוספת שדה/שדות נוספים שייצגו את סוג התו (האם מדובר בסוג תו שמנגן צליל "אמיתי", או תו-קשת / תו השתק), ובהתאם לדעת כיצד יש לפענח את ערך גובה הצליל והאם הוא מנוון או לאו, הוחלט פשוט "להתלבש" על מאפיין גובה הצליל ולאכלס בו ערך מיוחד לסיווג התו במקרה המיוחד שאכן מדובר בתו מיוחד. זוהי דרך אגב שיטת ייצוג מקובלת (קידוד ערכים מיוחדים לתווי השתק ותווי-קשת). למעשה, ההשראה למימוש זה נלקחה ממאמרים המתארים את המימוש של מערכת** [GenJam](https://igm.rit.edu/~jabics/BilesICMC94.pdf) **– מערכת המאלתרת מנגינות סולו בג'אז בזמן אמת בהופעות חיות באינטראקציה עם הנגנים על הבמה, תוך שימוש באלגוריתמים גנטיים ומאגר פרזות (מנגינות קצרות) מוכנות מראש.**

#### [NoteName Enumeration](https://github.com/cwelt/Soloist/wiki/NoteName)

**בכדי לזהות ולסווג את התווים את השונים ע"פ שם (דו, רה, מי,..., וכו'), הוגדר** Enumeration **המכיל את שמות כל התווים האפשריים במוסיקה במערכת הטונאלית. נעביר כי להבדיל מגבהי הצלילים, שבהם אנחנו מבחינים בין צלילים המייצגים אותו תו באוקטבות שונות (צלילים שהתדירות שלהם בהרץ היא ביחס של 1:2 בין אחד לשני באוקטבות עוקבות) ושסה"כ ישנם 128 כאלו ב-** MIDI, במיפוי השמי לתווים אנחנו לא מבחינים בגבהי צלילים שונים של אותו תו, לכן התו C למשל, מייצג את התו השייך לצלילים בכל האוקטבות **(בטבלה מעלה במיפוי של** MIDI **אלו הם כל הצלילים המסווגים תחת** C **ללא תלות במספר האוקטבה, למעשה כל הערכים שמתחלקים ב-12 ללא שארית: 0,12,24, וכו').**

**אוסף שמות זה שימושי כאשר נדרש לשייך צלילים לתווים בהקשר** (context**) נתון. עם שמות תווים ומיפוי שלהם לצלילים, ניתן למשל להגדיר בקלות שגרה שתחזיר את אוסף הצלילים** הרלוונטיים לאקורד מז'ור המושרש בתו דו (C), או אוסף הצלילים השייכים לסולם לה-מינור, כשעבור השורש מספיק לציין את שם התו, ולא את כל הגבהים הצלילים האפשריים הדבר קצת דומה למיפוי של DNS בין כתובת IP ל-Hostname. המיפוי השמי אינו הכרחי, אך **מאוד כדאי בכדי להפוך את החיים לקלים יותר.**

אוסף השמות מוגדר כ- Enumeration בקובץ NoteName.cs. אוסף מכיל את כל שמות התווים, כאשר כל תו מופיע תחת שלושה ערכים – האחד עבור התו "הנקי" עצמו, השני עבור התו עם סימן התק של במול (הורדה בחצי-טון) ש"פ המוסכמה מוגדר עם סיומת Flat המשורשרת לשם התו, ואחרון חביב השלישי, התו עם סימן התק של דיאז (העלאה בחצי-טון), שע"פ המוסכמה מוגדר עם סיומת Sharp המשורשרת לשם התו.

שמות התווים האפשריים הם כדלקמן –

|  |  |
| --- | --- |
| שם התו בכתיב עברי | שם התו בכתיב לטיני |
| לה | A |
| סי | B |
| דו | C |
| רה | D |
| מי | E |
| פה | F |
| סול | G |

לפיכך באוסף הקבועים מוגדרים השמות A, AFlat, ASharp, B, BFlat, BSharp וכן הלאה עד ל-G, כאשר כל תו מופיע כאמור בשלוש וריאציות: "נקי", עם תוספת "Flat" לסימון במול, ועם תוספת "Sharp" לסימון דיאז.

**אוסף זה למעשה מייצג את שמות התווים האפשריים במערכת** Soloist **ומשתתף במנשק באופן רוחבי בשכבת הלוגיקה העסקית, הן במסגרת הייצוג הישויות המוסיקליות (למשל בתור מאפיין תו השורש של אקורדים וסולמות) והן בהמרה בין ייצוג שונים של תווים מספריות חיצוניות, אלגוריתמי ההלחנה ועוד.**

#### [INote Interface](https://github.com/cwelt/Soloist/wiki/INote)

**המנשק** [INote](https://github.com/cwelt/Soloist/wiki/INote) **מייצג תו. תיעוד מלא ומפורט של המאפיינים שלעיל ניתן למצוא הן בגוף קוד המקור וכן ב**[דף ה-Wiki של המנשק](https://github.com/cwelt/Soloist/wiki/INote)**, 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.

המנשק מורכב מישויות שסקרנו קודם לכן – [משך שהייה](#_ייצוג_משך_שהייה), [גובה צליל](#_Enumeration_NotePitch), [ושם תו](#_Enumeration_NoteName).

#### [Note Class](https://github.com/cwelt/Soloist/wiki/Note)

**המחלקה** [Note](https://github.com/cwelt/Soloist/wiki/Note) **מממשת את המנשק** [INote](https://github.com/cwelt/Soloist/wiki/INote)**. מעבר למאפיינים והמתודות במנשק, מחלקה זו מגדירה גם בנאים, קבועים, ומתודות לאחזור שם התו ומחרוזת לתיאור התו. תיעוד מלא ומפורט של כלל הרכיבים במחלקה ניתן למצוא הן בגוף קוד המקור וכן** [בדף ה-Wiki של האפליקציה](https://github.com/cwelt/Soloist/wiki/Note)**. למען הסדר טוב מובא כאן תקציר התיעוד של הרכיבים הנוספים השייכים למחלקה עצמה (אלו שאינם חלק מהמנשק** [INote](#_INote_Interface)**) –**

בנאים: **בנאי מבוסס צליל ומשך שהייה, צליל ומונה/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: **ייצוג תו ע"י מחרוזת ואחזור 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) {…}

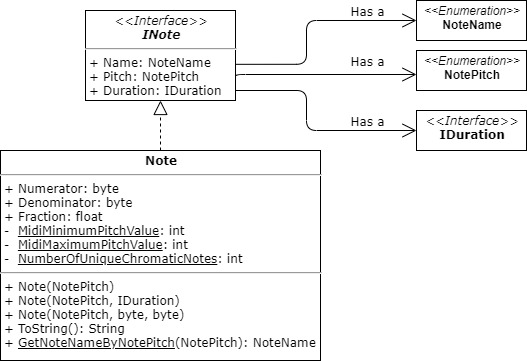
**קבועים:** קבועים אלו משמשים בבדיקות תקינות, ושגרות עזר במניפולציות על תווים –

private const int NumberOfUniqueChromaticNotes = 12;

private const int MidiMinimumPitchValue = 0;

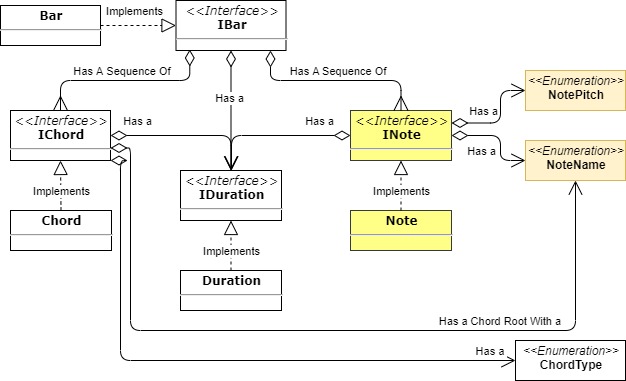
private const int MidiMaximumPitchValue = 127;

**דיאגרמת Class Diagram עבור הישויות לייצוג תו** –

****

**קשרים עם מחלקות אחרות**  –

רכיבי הייצוג של תו שהוצגו בפרק זה נמצאים בשימוש באופן רוחבי בשכבת הלוגיקה העסקית, שכן התווים מהווים את אבני הבניין של המנגינה וכל אלגוריתמי ההלחנה בסופו של דבר מבצעים מניפולציות על תווים אלו. להלן שוב דיאגרמת המחלקה של הישויות המוסיקליות המרכזיות בדגש על הקשר שלהן עם הישויות המייצגות תווים שהוצגו בסעיף זה –

****

### 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 באקורד מוגדרים ע"פ רצף מרווחים קבוע מראש מהשורש. רצף מרווחים קבוע זה מוגדר ע"פ סוג האקורד (מז'ור, מינור, וכד'). למשל: אקורד משולש מז'ורי מגדיר מרווח של ארבעה חצאי טונים מהשורש לצליל השני באקורד ולאחר מכן מרווח של שלושה חצאי טונים בין הצליל השני באקורד לצליל השלישי.

ביצירה מוסיקלית נתונה, האקורדים פרוסים לאורך היצירה וקובעים את המהלך ההרמוני שלה. אקורדים אלו מתנגנים לצד או "מאחורי" המנגינה, ומספקים לה בעצם את התשתית והליווי. בכדי להגדיר את המיפוי והסנכרון בין צלילים במנגינה לבין צלילי האקורדים המלווים אותם, גם לאקורדים יש משכי שהייה בדיוק כפי שלתווים מתנגנים יש משך שהייה. לפיכך, בכדי לייצג אקורד נדרשים שלושה מאפיינים – התו הנמצא בשורש האקורד, סוג האקורד, שקובע את רצף המרווחים מהשורש שמגדירים את יתר הצלילים שבאקורד, וכן משך שהייה, שקובע את זמן הנגינה של כל אקורד ביחד לתיבה שבה הוא מתנגן ביצירה מוסיקלית נתונה.

#### [ChordType Enumeration](https://github.com/cwelt/Soloist/wiki/ChordType)

**אוסף הקבועים ב-** [ChordType.cs](https://github.com/cwelt/Soloist/wiki/ChordType) **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 נלווים שמאחזרים את הצלילים/תווים הרלוונטיים בהתאם לסוג האקורד, וכן סט סוגי האקורדים מגדיר פורמט לסוגי האקורדים שמשתמשי קצה יכולים לציין בקבצי קלט שהם מעלים למערכת.**

#### [IChord Interface](https://github.com/cwelt/Soloist/wiki/IChord)

**המנשק** [IChord](https://github.com/cwelt/Soloist/wiki/IChord) **מייצג אקורד.**  אקורד מאופיין ע"י ישויות שסקרנו קודם לכן: [שם התו של שורש האקורד](#_Enumeration_NoteName) , [סוג האקורד](#_ChordType_Enumeration) ו[משך שהייה](#_ייצוג_משך_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.

**המנשק מגדיר** שני זוגות של מתודות עזר לאחזור אוספי תווים/צלילים ע"פ סוג האקורד: האחד מחזיר את הצלילים של תווים ממבנה האקורד עצמו, והשני מחזיר אוסף רחב יותר של צלילים מסולם שממופה מול סוג האקורד (המיפוי עצמו ממומש בשירות נפרד מחוץ למנשק). המתודות מאפשרות להעביר חסמים תחתונים ועליונים למנעד הצלילים שיחזור באוסף, ומוגדרים בשני העמסות, 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 שלעיל ניתן למצוא הן בגוף קוד המקור וכן ב**[**דף ה-**Wiki **של המנשק בספריית האפליקציה ב-**Github](https://github.com/cwelt/Soloist/wiki/IChord)**, המכיל קישורים לרכיבים השונים עם התיעוד שלהם.**

#### [Chord Class](https://github.com/cwelt/Soloist/wiki/Chord)

**המחלקה** [Chord](https://github.com/cwelt/Soloist/wiki/Chord) **מממשת את המנשק** [IChord](https://github.com/cwelt/Soloist/wiki/IChord)**. מלבד מימוש המאפיינים והמתודות של המנשק, ההגדרות הנוספות במחלקה כוללות רק בנאי ומתודה לאחזור מחרוזת המייצגת את האקורד.**

**תיעוד מלא ומפורט של כלל הרכיבים במחלקה ניתן למצוא הן בגוף קוד המקור וכן** [**בדף ה-**Wiki **של האפליקציה ב-**.Github](https://github.com/cwelt/Soloist/wiki/Chord) **למען הסדר טוב מובא כאן תקציר התיעוד של הרכיבים הנוספים השייכים למחלקה עצמה (אלו שאינם חלק מהמנשק** IChord**).**

בנאי: **המחלקה מגדירה בנאי יחיד ביוצר אקורד על סמך תו שורש, סוג אקורד ומשך-שהייה –**

// Constructs a new chord instance with the given root, chord type & duration.

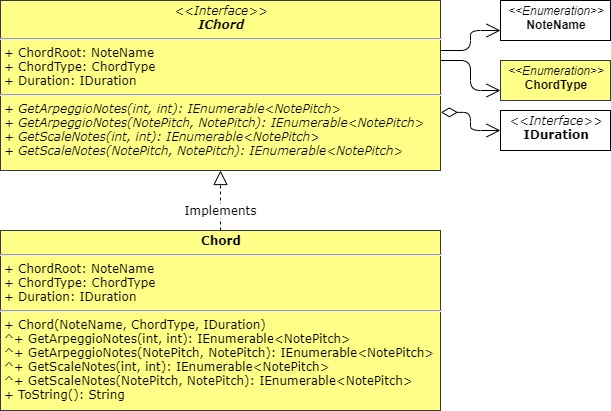
public Chord(NoteName root, ChordType type, IDuration duration)

מתודות: **המחלקה מגדירה מימוש של** ToString **לאחזור מחרוזת המייצגת אקורד –**

public override string ToString() =>

$"{{Root={ChordRoot}; ChordType={ChordType}; Duration={Duration}}}";

**דיאגרמת מחלקה** **:** להלן דיאגרמת מחלקה של הישויות המייצגות את האקורד שהוצגו בסעיף זה עם הקשרים ביניהן –

****

### תיבות (Bars)

**רקע**: יצירות מוסיקליות מחולקות לתיבות המסייעות בסגמנטציה וסדר ברצף הצלילים של היצירה. לכל תיבה מוגדר משך שהייה, בדיוק כמו שיש לתווים ולאקורדים משך שהייה. בהקשר של תיבה, משך שהייה 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 ההרמוני שמלווה את הצלילים הללו. תיבה יכולה גם להיות ריקה, ולהכיל רק אקורדים או תווי הפסקה, במקרה שבו יש מקטע של הפוגה במנגינה או בליווי ביצירה. משך השהייה הכולל של רצף אקורדים ו/או רצף תווים בתיבה נתונה לא יכול לחרוג ממשקל התיבה. במקרה שבו משך זמן השהייה של תו מתנגן חוצה שתי תיבות או יותר, המעבר בכל תיבה עוקבת מסומן עם תו קשת-חיבור, שמאריך את משך השהייה, כך שבגוף כל תיבה אכן אין חריגה מהמשקל שלה.

בכדי לייצג תיבה, נדרשים איפה שלושה מאפיינים: חותמת הזמן של התיבה, אוסף התווים שהיא מכילה (תווי המנגינה), ואוסף האקורדים שבתיבה. כמו כן, מאחר שהאקורדים מלווים את תווי המנגינה, יש צורך במנגנון של סנכרון בזמנים בכדי לדעת בנקודת זמן נתונה שבה מתנגן תו איזה אקורד מתנגן מאחוריו בליווי ולהפך: איזה תווים מתנגנים באותו מחזור זמן של אקורד נתון בתיבה. סנכרון זה נדרש בכדי להתאים תווים לאקורדים בהלחנת מנגינות חדשות.

#### [IBar Interface](https://github.com/cwelt/Soloist/wiki/IBar)

**המנשק** [IBar](https://github.com/cwelt/Soloist/wiki/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 המתודות שתוארו לעיל המוגדרות במנשק –**

// 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 והמתודות שלעיל ניתן למצוא הן בגוף קוד המקור וכן ב**[**דף ה**-Wiki **של המנשק**](https://github.com/cwelt/Soloist/wiki/IBar)**, המכיל קישורים לרכיבים השונים עם התיעוד שלהם.**

#### [Bar Class](https://github.com/cwelt/Soloist/wiki/Bar)

**המחלקה** [Bar](https://github.com/cwelt/Soloist/wiki/Bar) **מממשת את המנשק** [IBar](https://github.com/cwelt/Soloist/wiki/IBar)**. מעבר למימוש המאפיינים והמתודות של המנשק, מחלקה זו מגדירה רק בנאים ומתודה סטנדרטית לאחזור מחרוזת המייצגת את התיבה.**

**תיעוד מלא ומפורט של כלל הרכיבים במחלקה ניתן למצוא הן בגוף קוד המקור וכן** [בדף ה-Wiki של האפליקציה](https://github.com/cwelt/Soloist/wiki/Class)**. למען הסדר טוב מובא כאן תקציר התיעוד של הרכיבים הנוספים השייכים למחלקה עצמה (אלו שאינם חלק מהמנשק** [IBar](#_IBar_Interface)**).**

שדות: **המחלקה מגדירה שני שדות פרטיים הנגזרים מהמונה 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.

בנאים: **המחלקה** Bar **מגדירה 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)

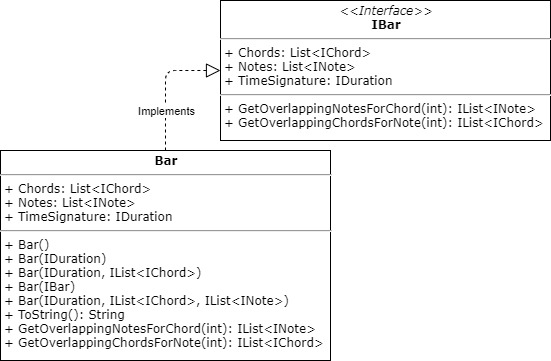
// Copy constructor: notes & durations deep copy, chords is shallow copy.

public Bar(IBar bar) {…}

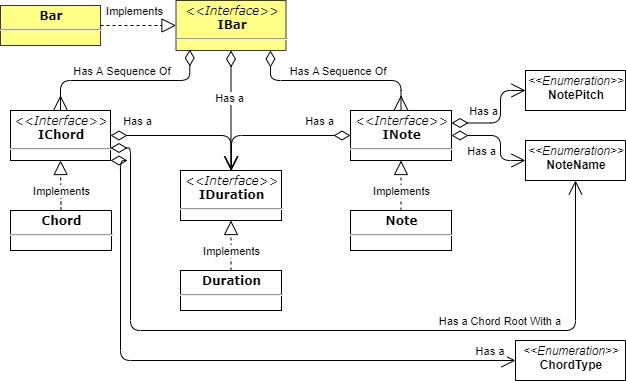
מתודות: **המחלקה מגדירה מימוש של** ToString **לאחזור מחרוזת המייצגת תיבה –**

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

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



להלן דיאגרמת מחלקה של רכיבי המנשק והמחלקה המייצגים תיבה והקשר שלהן בקונטקסט עם יתר רכיבי הפיתוח של הישויות המוסיקליות –

****

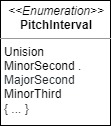
### מרווחים (Intervals)

**רקע**: מרווח הוא מרחק בין גבהי צלילים, הנמדד בחצאי-טונים, זהו המרווח הקטן ביותר במוסיקה המערבית – המרווח בין כל שני קלידים בפסנתר (וכן כל שני סריגים בצוואר גיטרה). למרווחים יש משמעות רבה בהלחנת מנגינות – למשל, מרווח גדול במיוחד גורם לקפיצה לקיצונית מצליל גבוה לנמוך (או להפך), מרווחים מסוימים נוטים להישמע זכים (מתוקים) בקונטקסט מסוים בעוד שאחרים יוצרים מותח ודיסוננס. לפיכך, יש צורך בייצוג נאות של מרווחים לטובת אלגוריתמי ההלחנה.

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

#### [PitchInterval Enumeration](https://github.com/cwelt/Soloist/wiki/PitchInterval)

אוסף זה נמצא בשימוש בעיקר ע"י אלגוריתמי ההלחנה, שמשתמשים במרווחים בכדי לאמוד מדדים שונים באשר טיב המעברים בין צליל אחד למשנהו (למשל סיווג מרווח שהחל ממנו המעבר הוא חד מידי) וכן בחישובים אריתמטיים של התווים השכנים תוך שימוש במרווחים.



internal enum PitchInterval

{

Unison = 0,

MinorSecond = 1,

MajorSecond = 2,

…

Octave = 12

}

### מחלקת מפעל לישויות מוסיקליות (MusicTheoryFactory)

**מחלקה זו היא מחלקת שירות סטטית המספקת מתודות שירות ליצירת מופעים של מחלקות קונקרטיות שמממשים את מנשקי הישויות המוסיקליות השונות בתת-שכבה זו של התאוריה המוסיקלית – משך שהייה (**[IDuration](#_IDuration_Interface)**), תו (**[INote](#_INote_Interface)**), אקורד (**[IChord](#_IChord_Interface)**) ותיבה (**[IBar](#_IBar_Interface)**), ומכאן שם המחלקה (על שם דפוס העיצוב** FactoryMethod**).**

**המחלקה כוללת ארבעה קטגוריות של מתודות יצירה שונות – קטגוריה אחת כנגד כל ישות מוסיקלית, כאשר בכל קטגוריה המתודות הן בעלות אותו שם ונבדלות רק בסט הארגומנטים שלהן.**

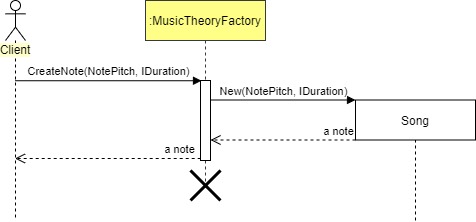
**המוסכמה (קונבנציה) של שמות המתודות במחלקה זו היא** "Create<Entity>"**, כאשר** <Entity> **מחוץ בשם הישות (ללא אות הראשית** I **משם ה-**Interface**). למשל עבור המנשק** INote **המייצג ישות של תו מוסיקלי, המחלקה תגדיר מתודות יצירה בשם** "CreateNote"**. להלן הישויות המנוהלות כרגע תחת מחלקת** Factory **זו –**

|  |  |  |
| --- | --- | --- |
| Method Name | Entity | **#** |
| CreateDuration | [IDuration](#_IDuration_Interface) | **1** |
| CreateNote | [INote](#_INote_Interface) | **2** |
| CreateChord | [IChord](#_IChord_Interface) | **3** |
| CreateBar | [IBar](#_IBar_Interface) | **4** |

**המוטיבציה מאחורי מחלקה זו היא לשבור את התלות של קליינטים במימוש ספציפי של ישות מוסיקלית כזו או אחרת, ולהשאיר תלות רק באבסטרקציה, דהיינו רק במנשק עצמו. קליינטים המעוניינים במופע של ישות מוסיקלית יפנו למחלקת ה-**MusicTheoryFactory **בבקשה לקבלת מופע של המנשק, והמחלקה תהיה אחראית לספק מופע שכזה, וכל זאת בלי שהקליינטים צריכים להכיר את המחלקות שמספקות את המימוש של המנשק בפועל מאחורי הקלעים. זהו בעצם מימוש של עיקרון ה-**[Dependency Inversion Principle](https://en.wikipedia.org/wiki/Dependency_inversion_principle)**.**

**נכון לעכשיו, המימוש של ה-**Factory **כולל פשוט הפנייה לבנאים של המחלקות המוגדרות בתת-שכבה זו הממשות מנשקים אלו, שכן זהו המימוש היחידי המוגדר כעת במערכת, אולם אם בעתיד יהיה צורך בהחלפה של מימושי ברירת מחדל אלו במימושים משוכללים יותר, מלבד הוספת החתימות החדשות במנשקים עצמם** )**במידת הצורך) כל שיידרש הוא לבצע את ההחלפה במימוש הפנימי של מתודות ה-**Factory**, כלומר, לקליינטים עצמם ההחלפה תהיה "שקופה", הם ימשיכו לפנות כרגיל אל ה-**Factory **בבקשה לקבלת מופעים של ישויות מוסיקליות ויקבלו מופעים של המנשקים שלהן, מבלי בכלל לדעת מהי המחלקה הקונקרטית שמספקת את המימוש.**

**להלן המחשה של אופן העבודה של קליינטים עם מחלקת ה-**Factory**. בדוגמה זו הקליינטים מבקשים מה-**Factory **מופע של** INote**, זו בתורה יוצרת מופע של המחלקה** Note **המממשת את המנשק** INote **ע"י קריאה ישירה לבנאי של המחלקה, ומחזירה את התוצאה לקליינט. מנקודת המבט של הקליינט, הוא ביקש** INote **וקיבל את מבוקשו. אין זה מעניינו איזו מחלקה ספציפית עומדת מאחורי המימוש, כך שבעתיד, אם יוחלף המימוש של** Note **במחלקה אחרת, הדבר יהיה שקוף לקליינטים העובדים ישירות מול המנשק** INote **–**

****

### הרחבות ושירותים נוספים

**רקע**: מעבר למנשקים, מחלקות ואוספי קבועים שהוצגו לעיל, המייצגים את הישויות המוסיקליות, הוגדרו בנוסף שתי מחלקות סטטיות (מחלקות שירות) הכוללות הרחבות ושירותי עזר סביב הישויות המוסיקליות השונות – [**MusicTheoryExtensions**](#_MusicTheoryExtensions) ו- **MusicTheoryServices**.

#### MusicTheoryExtensions

מחלקה זו כשמה כן היא, אוסף של כל ה-Extension Methods עבור הישויות המוסיקליות השונות המספקות שירותים משלימים ששימושיים בעיקר עבור אלגוריתמי ההלחנה. מחלקה זו מכילה אך ורק Extension Methods ללא מתודות סטטיות נוספות.

שאלה מתבקשת היא מדוע שירותים אלו הוגדרו במחלקה נפרדת זו כמתודות סטטיות ולא במנשקים והמחלקות של הישויות המוסיקליות עצמן כמתודות מופע ישירות. ובכן, הסיבה המרכזית שהשירותים לא הוגדרו במנשקים היא מאחר ששירותים אלו רלוונטיים בעיקר לאלגוריתמי ההלחנה, ובקונטקסט הרחב יותר של ישויות מוסיקליות שלאו דווקא עוסק בהלחנה אלגוריתמית, שירותים אלו "יזהמו" את החוזים השונים המתארים את הפונקציונאליות המצופה מכל ישות וישות, ללא תלות בהלחנה. הסיבה שמתודות אלו לא הוגדרו ישירות במחלקות כמימושים מחוץ למנשק היא שחלק מהשירותים ממילא ניתנים לאוסף של ישויות ולא על ישות בודדת ולכן ממילא יש להגדירם כמתודה סטטית (מתודת Extension גם היא סוג של מתודה סטטית מבחינת ההגדרה התחבירית שלה), וסיבה נוספת היא ריכוזיות – יש יתרון בריכוז כל השירותים הנלווים הללו במחלקה אחת, כך שאם בעתיד יוחלף מימוש של אחד המנשקים של הישויות המוסיקליות במימוש אחר/חדש ע"י מחלקה אחרת, השירותים הנלווים שסופקו ע"י המחלקה המרכזית לא "ילכו לאיבוד" וימשיכו לתת שירות ללא תלות במימוש הספציפי ע"י מחלקה כזו או אחרת.

להלן המתודות המספקות את השירותים המרוכזים במחלקה זו –

|  |  |  |
| --- | --- | --- |
| Description | The Subject Entity | Method |
| Returns a flattened sequence of all note in the given bar sequence / bar. | IEnumerable<IBar> | GetAllPitches |
| IBar | GetAllPitches |
| Return indication whether the given note in the given context is played on an offbeat or not. | INote | IsOffBeatNote |
| IBar | IsOffBeatNote |
| 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. | INote | GetDurationInContext |
| Returns the preceding note of a given note in the given note sequence. | IEnumerable<IBar> | GetPredecessorNote |
| Returns the succeeding note of a given note in the given note sequence. | IEnumerable<IBar> | GetSuccessorNote |
| Returns a simplified duration by reducing the numerator and denominator by their greatest common divisor (gcd), for example reduce 4/8 to 1/2. | IDuration | ReduceFraction  ToLowestTerms |

#### MusicTheoryServices

מחלקה זו מכילה כל שירות משלים או נוסף הנוגע לישויות המוסיקליות שאינו מוגדר ישירות במנשקים ובמחלקות של הישויות עצמן וכן אינו מוגדר כמתודת Extension של ישויות אלו או של אוספים שלהן. שירותים אלו כוללים השלמת פונקציונאליות קריטית שחלק מהמחלקות צורכות ולמעשה תלויות בו לטובת מימוש המנשקים.

המוטיבציה הראשונית להגדרת שירותים אלו במחלקה ייעודית נפרדת היא שחלק מהמימושים הנוכחיים שלהם תלויים בספריות צד ג'. מתוך רצון שליבת המודול של התאוריה המוסיקלית תהייה עצמאית ונקייה מתלויות ישירות בספריות צד ג', כל האינטראקציה של ישויות התאוריה המוסיקלית התלויה בספריות חיצוניות צד ג' נגרעה מתוך המחלקות המייצגות את הישויות עצמן ורוכזה במחלקה זו. מחלקה זו בעצם מהווה מעין חוצץ המושפע מהרעיונות של דפוסי העיצוב של Bridge ו-Adapter, שקוטע את ה-coupling של מחלקות הליבה מהספריות החיצוניות. למעשה, כל יתר המחלקות והמנשקים של ישויות התאוריה המוסיקלית כלל לא מכירים את הספריות צד ג'. אלו מהן שכן נדרשים בהשלמת השירותים הנוספים פונים בבקשה למחלקה זו שאחראית לספק אותם בעצמה תוך שימוש פנימי ע"פ הצורך בשירותי עזר של הספריות החיצוניות, באופן ששקוף לישויות הליבה שמשתמשים בשירותים אלו.

כשכל הקוד הנוגע לישויות המוסיקליות של הספריות החיצוניות מרוכז במחלקה אחת, יהיה קל מאוד בהמשך לבדוק את התלויות בספריות אלו ולהחליף אותן ע"פ הצורך בספריות אחרות או אף במימוש עצמי. למעשה התלות הנוכחית היא די דלה, ועם טיפה עבודה אפשר לממש בקלות את השירותים הללו עבור ישויות התאוריה המוסיקלית, ואחד מקווי המחשבה בניסיון להגיע ל-decoupling היה באמת לממש שירותים אלו באופן עצמאי מ-"scratch", אולם במחשבה נוספת הגיעה התובנה שדווקא יש לעשות reuse במה שיש ולא להמציא את הגלגל מחדש רק בשביל אי-תלות, לכן כן הוחלט לעשות שימוש בשירותים הקיימים אך תוך בידולם וריכוזם למחלקה ייעודית, שתבצע המרות דרושות בין הייצוגים של הישויות בספרייה האפליקציה ובין הייצוג המקביל שלהן בספריות צד ג', ותהווה מעין חוצץ-מתאם ששובר את התלות של הישויות בקוד מספריות חיצוניות.

להלן המתודות המספקות את השירותים המרוכזים במחלקה זו –

|  |  |
| --- | --- |
| Description | Method |
| Returns a note name for a given note pitch. | GetNoteName |
| Converts a note name from its external third-party library representation into its equivalent internal representation note name and vise-versa (from external to internal). These are required for using the external third-party services. | ConvertToInternalNoteName |
| ConvertToExternalNoteName |
| 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. | GetNotes |
| Returns a new duration instance which is the result of the given arithmetic operation, and the two is given duration operands. | DurationArithmetic |

בנוסף למתודות אלו המחלקה מגדירה גם קבועי עזר – **SemitonesInOctave** העוטף את "מספר הקסם" 12, שמייצג את מספר הצלילים בסולם הדיאטוני המערבי, וה-Enum הנקרא **ArithmeticOperation** המכיל קבועים עבור הפעולות האריתמטיות המוגדרות על אופרנדים מסוג משך שהייה (IDuration).

## תת-שכבת שירותי מידי (MIDI)

מרחב שמות (Namespaces) רלוונטי: CW.Soloist.CompositionService.Midi

### רקע – פרוטוקול MIDI וקובצי SMF

* **פרוטוקול מידי**: MIDI (Musical Instrument Digital Interface) הוא פרוטוקול לשיתוף והעברה של אותות ופקודות מוזיקליות בין כלי נגינה אלקטרוניים (דוגמת סינטיסייזר) לרבות כלי נגינה וירטואליים המנוהלים בתוכנת מחשב. פקודות מוזיקליות אלו מכילות הוראות לכלי הנגינה האלקטרוניים (והווירטואליים) על איזה צלילים עליהם לנגן ואיזה צלילים להפסיק לנגן, באיזו עוצמה יש לנגן כל צליל, מהו כלי הנגינה הווירטואלי שאחראי לבצע את הניגון ועוד. בטרמינולוגיה של MIDI, הודעות ופקודות אלו קרויים אירועים (Events). על-כן נשתמש במושג זה בהמשך במקום הודעות/פקודות/אותות.
* **קובצי מידי**: הפורמט הנפוץ להעברת אירועי MIDI בתוכנות מחשב הוא קובץ מידי סטנדרטי עם סיומת .mid, למשל "mySong.mid". קבצים אלו מוגדרים בסטנדרט [SMF (Standard MIDI File)](http://www.music.mcgill.ca/~ich/classes/mumt306/StandardMIDIfileformat.html#BM2_2), והם פופולריים מאוד בשימוש בתוכנות עריכה של מוסיקה, מאחר שלהבדיל מפורמטים נפוצים אחרים של קבצי שמע כגון mp3 או wav שמכילים ממש audio, קבצי המידי אינם מכילים audio כלל, אלא רק תיאורי מוסיקלי שלו ע"י אוספי אירועי MIDI הכוללים פקודות שעל כלי הנגינה האלקטרוניים והווירטואליים לפענח ולבצע בשביל להפיק את הנגינה המתבקשת.
* **יתרונות וחסרונות המידי:** שיטה זו של קידוד האירועים במקום את הקלטת ה-audio עצמה אמנם אינה מושלמת, וכוללת חסרונות מסוימים – כפי שלדפדפנים יש חופש מסוים באופן הניתוח, פענוח והצגה של דפי HTML, כך גם לתוכנות שמע שונות יש מרווח מסוים של חופש במימוש הסטנדרט, כך שתוכנות שונות עלולות לפענח את אירועי המידי טיפ טיפה אחרת אלו מאלו במובנים של אפקטים של כלי הנגינה, [קוונטיזציה](https://en.wikipedia.org/wiki/Quantization_(music)) של קבוצות צלילים ועוד, כך שתוצר השמע הסופי עשוי להישמע טיפה אחרת כאשר הוא מתנגן ע"י תוכנות שמע שונות. חיסרון נוסף הוא שהסאונד הסופי הוא מלאכותי (וירטואלי), ולכן ואף שהיום הטכנולוגיה מאפשרת להתחקות אחרי הניגון האנושי באופן די מדויק, הביצוע של הניגון המלאכותי לא מכיל את כל הדקויות של הנגן האנושי כך שהתוצר הסופי לא בהכרח משקף את כוונת המשורר (או ליתר דיוק –כוונת המלחין/מחבר הקובץ). עם זאת, לקובצי המידי יתרונות משמעותיים מאוד –

1. חיסכון בזיכרון: בעוד שקובצי שמע שמכילים audio כגון mp3 ו-wav מגיעים בקלות למשקל שנע בין כמהMB בודדים לעשרות ואף מאות MB, קובצי מידי טיפוסיים צורכים רק כמה עשרות או מאות בודדות של KB, לכן קל לאחסן ולאחזר אותם ולהעבירם ברשת על פני ערוץ תקשורת בין שרת ללקוח ובין מחשב לכלי נגינה אלקטרוני.
2. הקבצים "פתוחים" או "חשופים" במובן שהם מכילים רשימה של כל האירועים המוזיקליים שעל כלי הנגינה האלקטרוניים והווירטואליים להגיב אליהם על מנת לנגן את המוסיקה המתוארת בקובץ. לפיכך, ניתן לחלץ את ה-"data" מתוך הקובץ, לערוך אותו, ולשמור אותו בחזרה. קיימות מגוון תוכנות בשוק, לרבות תוכנות בקוד-פתוח, התומכות בייבוא וייצוא של קובצי מידי, שמספקות שירות של קריאת תוכן האירועים ועיבוד שלהם, דוגמת פענוח האירועים והצגת רצף התווים שהם מייצגים, שינוי גובה הסולם של האירועים, המרת המידי לקובץ audio, ועוד. למעשה, קובצי המידי כל כך פופולריים בתעשייה היום שכל תוכנה בתחום עריכה וניגון של תווים שמכבדת את עצמה מוכרחת לספק תמיכה בקריאה וכתיבה של תוכן "מ-" / "אל" קבצי מידי.

* **מבנה כללי של קובצי המידי:** קובצי המידי מחולקים לבלוקים (Chunks) כאשר אחד מהם הוא בלוק כותרת (Header Chunk) הכולל אירועי metadata על הקובץ, דוגמת תיאור טקסטואלי של שם הקטע המוסיקלי, מאפייני מקצב וטמפו (BPM) וכד', ואילו יתר הבלוקים מכילים אירועים רצועות (Track Chunks). לרוב כל רצועה מכילה אירועים המיועדים לכלי נגינה יחיד. בכדי לממש זאת, המידי מנהל ערוצים (channels), כך שכאשר נשלח אירוע לערוץ מסוים, רק רכיבים שמאזינים לאירועים על אותו ערוץ מגיבים לאירועים אלו. אפשר להסתכל על הערוץ כמושג כאנלוגי למושג ה-port ב-socket-ים בתקשורת. בהקשר של תוכנת שמע, הרכיבים הם כלי נגינה וירטואליים שונים, למשל על ערוץ 1 מאזין פסנתר, ערוץ 2 גיטרה, וכן הלאה. הסטנדרט תומך בעד 16 ערוצים שונים. קיים גם פורמט למידי המכיל את כל האירועים על רצועה אחת ויחידה (Format 0). האופי המונוליטי של פורמט זה מסבך מאוד את הניתוח ועריכה של אירועים אינדיווידואליים, על כן יישום זה תומך רק בפורמט מידי שתומך בריבוי רצועות (Formats 1-2).
* **קובצי מידי כקובצי שמע:** אמנם קובצי מידי כאמור אינם מכילים אודיו, אלא רק אוסף אירועים, אך לאור הפופולריות הרבה של ה-MIDI, מרבית יישומי נגני המוסיקה ב-PC ומכשירים סלולאריים (Media Player, VLC, Winamp וכד') תומכים בקבצי מידי, ויודעים לפענח אותם ולנגן את אירועי המידי שבקבצים אלו על כלי נגינה וירטואליים.

### ארכיטקטורת תת-שכבת שירותי ה-MIDI

להבדיל מפלטפורמות אחרות (למשל [Java](javax.sound.midi)) שמספקות ספריות סטנדרטיות מובנות לתמיכה בפרוטוקול MIDI, פלטפורמת ה-.NET לא מספקת ספריות שכאלה. עם זאת, ישנן מספר ספריות ופרויקטים בקוד-פתוח שניתן לעשות בהן שימוש, חלקן מיועדות לתכנות מול כלי נגינה אלקטרוניים כמו סינטיסייזר ומספקות רק תמיכה בעבודה ישירה מול אירועי המידי ע"פ הפרוטוקול עצמו (API ברמת low-level שדורש היכרות מעמיקה עם פרטי הפרוטוקול), ואחרות עוטפות את הפרטים הטכניים של הפרוטוקול ומספקות ממשק API ברמת High-Level שמבצע אבסטרקציה על הפרוטוקול ומאפשר לקליינטים לדבר בשפה של ישויות מוסיקליות כמו תווים ולא בשפה של אירועי מידי.

מערכת Soloist עושה שימוש בספרייה הבאה, שמאפשרת עבודה מול מידי גם ברמת האירועים ב-low-level וגם ברמת הפשטה גבוהה יותר high-level בהתאם לצורך –

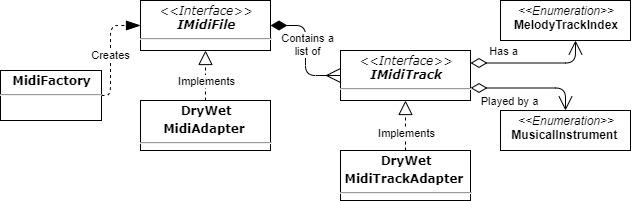
<https://github.com/melanchall/drywetmidi>

תת-שכבת שירותי המידי מסתירה את פרטי המימוש משאר תתי-השכבות והשכבות האחרות במערכת, ובפרט אינה חושפת את שירותי הספרייה החיצונית שלעיל, אלא מכמיסה אותה ועוטפת אותה במחלקת Adapter שממירה את השירותים הנדרשים ממנה לשירותים שיתר השכבות מכירות במנשק IMidiFile שיפורט לעיל, כך שאם בעתיד יוחלט להחליף את מימוש שירותי ה-MIDI בספרייה אחרת (למשל אם Microsoft יפתחו בעתיד ספרייה משלהם לתמיכה בפרוטוקול MIDI), כל שיידרש הוא להגדיר מתאם Adapter חדש בין הספרייה ל-Interface, ולעדכן את הזרקת התלות (Dependency Injection) של הקליינטים עם ה-Adapter המעודכן.

להלן רכיבי הפיתוח המוגדרים בתת-שכבה זו –

|  |  |  |  |
| --- | --- | --- | --- |
| # | תפקיד | סוג רכיב פיתוח | שם הרכיב |
| 1 | מספר רצועת המנגינה בקובץ ה-MIDI | Enumeration | [MelodyTrackIndex](#_אוסף_מספרי_רצועות) |
| 2 | ייצוג כלי נגינה וירטואליים ב-MIDI | Enumeration | [MusicalInstrument](#_MusicalInstrument_Enumeration) |
| 3 | רצועת כותרת/נגינה בקובץ MIDI | Interface | [IMidiTrack](#_מנשק_רצועת_מידי) |
| 4 | Class | [DryWetMidiTrackAdapter](#_מחלקת_אדפטר_לרצועת) |
| 5 | קובץ MIDI | Interface | [IMidiFile](#_IMidiFile_Interface) |
| 6 | Class | [DryWetMidiAdapter](#_מחלקת_אדפטר_לקובץ) |
| 7 | יצירת קובצי MIDI | Class | [MidiFactory](#_מחלקת_מפעל_לישויות_1) |

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



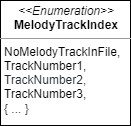
### אוסף מספרי רצועות (MelodyTrackIndex Enumeration)

**כאמור, אחד היתרונות הגדולים בקובצי מידי הוא שניתן לקרוא את תיאור האירועים המוסיקליים, לעבד אותם תוך הפעלת לוגיקה עסקית ולבסוף לכתוב בחזרה את האירועים המעודכנים וע"י כך להפיק קובץ מידי חדש/מעודכן שניתן להשמיע בנגני מוזיקה.**

**בהקשר של אפליקציה זו של הלחנת מנגינות חדשות על בסיס יצירות קיימות, האפליקציה צריכה לספק יכולת להחליף מנגינה קיימת. בקובצי מידי ניתן לבצע בזאת בקלות ע"י החלפת הרצועה שמכילה את אירועי המנגינה הקיימת ברצועה חדשה שמכילה את אירועי המגינה המעודכנת. בכדי שהאפליקציה "תדע" איזו רצועה יש להחליף (דהיינו איזו רצועה מכילה את המגינה המקורית), על המשתמש לספק את מספרה הסידורי: האפליקציה העצמה אינה יכולה להסיק נתון זה בעצמה שכן בקובץ מידי טיפוסי ישנם מספר רצועות המכילים אירועים של ניגון תווים. אמנם, ניתן לבצע איזשהו פילוח והשוואה בין הרצועות ולבדוק איזו מהרצועות מכילה אירועים של תווים היוצרים מנגינת יחיד, אולם יש גם מקרים מורכבים במיוחד שבו המנגינה חוזרת על עצמה בכמה רצועות שונות בקולות שונים (למשל במרווח של טרצה), על-כן בכל מקרה החלטה זו של איזו מהמנגינות יש להחליף צריכה להגיע מהמשתמש.**

**פרוטוקול** MIDI**תומך בעד 16 ערוצים, ובד"כ, לפחות במוסיקה קלה, להקה או הרכב כוללים רק כמה נגנים בודדים (פחות מ-16) כך שכל תפקיד (כלי נגינה) כך שניתן להקצות לכל כלי נגינה/תפקיד רצועה נפרדת עם ערוץ שידור משלה. נקודת ההנחה היא איפה שקובץ המידי יכיל 16 רצועות לכל היותר (בהתאם למגבלות 16 הערוצים). בכדי לאכוף אילוץ זה בקלות תוך הגדרה בעלת משמעות ברורה, הוגדר בקובץ** MelodyTrackIndex.cs **אוסף קבועים עבור אינדקסים של 16 רצועות אפשריות בקובץ, כאשר הספירה מתחילה ב-1** (one-based indexing). **כמו כן, הוגדר קבוע אחד לייצוג המקרים שבהם קובץ המידי הוא פלייבק "טהור" שמכיל רק את הליווי ללא מנגינה, כלומר המקרים שבהם נדרש לשבץ מנגינה בקובץ ברצועה חדשה מבלי להחליף רצועה קיימת.**

**להלן ההגדרה הבסיסית של אוסף הקבועים –**



public enum MelodyTrackIndex

{

NoMelodyTrackInFile = 0,

TrackNumber1 = 1,

TrackNumber2 = 2,

…

TrackNumber16 = 12

}

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

### אוסף כלי נגינה (MusicalInstrument Enumeration)

**ב-**MIDI **מוגדרת רשימת קודים כללים (**[General MIDI Program Change Instruments](https://www.midi.org/specifications-old/item/gm-level-1-sound-set)**) המייצגים 128 כלי נגינה ו/או אפקטים שונים, המחולקים למשפחות – פסנתרים, כלי נשיפה, כלי מיתר, כלי קשת ועוד.**

**ע"י שימוש בערכים אלו באירועים המתאימים בפרוטוקול 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 החדשה. בכדי למנוע שימוש ב-"ערכי קסם" קבועים שמייצגים את כלי הנגינה והאפקטים השונים, הרשימה הוטמעה במערכת** Soloist **כאוסף קבועים (**Enumeration) **בקובץ** MusicalInstrument.cs. **אמנם רשימה זו כבר מוגדרת בספרייה החיצונית** [DryWetMidi](https://github.com/melanchall/drywetmidi) **שהמערכת עושה בה שימוש, אולם לאור הרצון לתחום ולכמוס את כל פרטי הספריות החיצונית מיתר שכבות המערכת, הוחלט להגדיר רשימה זו מחדש כך שלא תהיה תלויה בספרייה החיצונית שבה נעשה שימוש לאספקת שירותי ה-**MIDI**. כמו כן, הרשימה שבה נעשה שימוש במערכת היא רשימה חלקית בלבד: היא מגדירה רק את 112 הערכים הראשונים ואינה תומכת (במכוון) ביתר 16 הערכים של משפחות כלי ההקשה אפקטים, שאינם רלוונטיים לניגון מנגינות.**

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

**להלן ההגדרה הבסיסית של אוסף הקבועים של כלי הנגינה –**



public enum MusicalInstrument

{

AcousticGrandPiano,

BrightAcousticPiano,

ElectricGrandPiano,

…

}

### מנשק רצועת מידי (IMidiTrack Interface)

**המנשק** IMidiTrack **מייצג רצועה מוסיקלית בודדת בקובץ מידי. הוא מתפקד כמבנה בלבד (**POCO – Plain Old CLR Object**) ללא פונקציונאליות, המכיל מקבץ של מאפיינים רלוונטיים ו-"מעניינים" של רצועה מוסיקלית בקובץ מידי: מספרה הסידורי של הרצועה בקובץ המידי (אינדקס), שם הרצועה, קוד כלי הנגינה הווירטואלי שמבצע את אירועי המידי של רצועה זו, ושם כלי הנגינה הווירטואלי המתאים לקוד כל הנגינה שלעיל –**

|  |  |  |
| --- | --- | --- |
| Description | Type | Property |
| One-based ordinal number of the track in its containing midi file. | [MelodyTrackIndex](#_MelodyTrackIndex_Enumeration) (Enumeration) | TrackNumber |
| Midi sequence track name. | string | TrackName |
| The [general MIDI program number](https://en.wikipedia.org/wiki/General_MIDI) which identifies this track's MIDI musical instrument. | [MusicalInstrument](#_MusicalInstrument_Enumeration)  (Enumeration) | InstrumentMidiCode |
| The instrument description that corresponds to the instrument code. | string | InstrumentName |

### מנשק קובץ מידי (IMidiFile Interface)

המנשק IMidiFile מייצג קובץ מידי סטנדרטי (SMF) בעל סיומת mid. מנשק זה הוא מנשק "lightweight", במובן שהוא אינו מכיל את כלל התכונות והפונקציונאליות שהיינו מצפים שיהיו במנשק המייצג ישות של קובץ מידי, אלא רק את מה שהכרחי ליישום המערכת. האינטראקציה המלאה מול קבצי המידי מבוצעת כאמור באמצעות שירותים של הספרייה החיצונית [DryWetMidi](https://github.com/melanchall/drywetmidi), אולם הגישה אליה היא רק באמצעות מחלקת Adapter ייעודית (פירוט בהמשך) שמממשת את המנשק IMidiFile Interface שאותו יתר השכבות מכירות ודרכו הן צורכות את שירותי ה-MIDI של תת-שכבה זו.

להלן המאפיינים (Properties) המוגדרים במנשק זה –

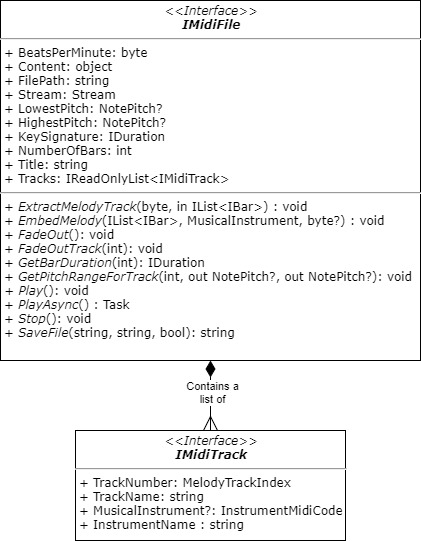
|  |  |  |
| --- | --- | --- |
| **Description** | **Type** | **Property** |
| Absolute physical path of the MIDI file. | string | FilePath |
| Input source stream which is used to read the midi content from. | Stream | Stream |
| The actual content the MIDI file. | object | Content |
| MIDI Sequence Name from the header track meta events. | string | Title |
| [BPM](https://bit.ly/2LIuuVE) –Beats Per Minute. The tempo (speed) that is set for playing the events in the MIDI file. | byte | BeatsPerMinute |
| Total number of bars in the MIDI file. | int | NumberOfBars |
| The key signature from the MIDI meta events. | [IDuration](#_IDuration_Interface_1) | KeySignature |
| Global lowest pitch in the entire midi file. | [NotePitch](#_NotePitch_Enumeration)? | LowestPitch |
| Global highest pitch in the entire midi file. | [NotePitch](#_Enumeration_NotePitch)? | HighestPitch |
| The tracks (chunks) contained in the MIDI file. | IReadOnlyList<[IMidiTrack](#_IMidiTrack_Interface)> | Tracks |

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

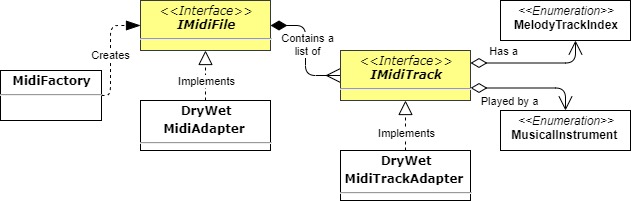
להלן תקציר השיטות (Methods) המוגדרות במנשק זה –

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

**להלן דיאגרמת מחלקה עבור המנשקים של רצועת MIDI וקובץ MIDI** –



**להלן דיאגרמה המדגישה את הקשר שלהן ליתר רכיבי הפיתוח במודול ה-MIDI** –



### מחלקת מעטפת לקובץ מידי (DryWetMidiAdapter Class)

**המחלקה** DryWetMidiAdapter **מממשת את המנשק** [IMidiFile](#_IMidiFile_Interface)**, תוך שימוש בספריית עזר חיצונית –** [DryWetMidi](https://github.com/melanchall/drywetmidi), **שאותה היא עוטפת** **ומסתירה (מכמיסה) וחושפת כלפי חוץ את המנשק הפשוט** [IMidiFile](#_IMidiFile_Interface) **שלו מצפים הקליינטים, ומכאן שם המחלקה (על שם דפוס העיצוב** Adapter**).**

**הספרייה החיצונית אמנם מספקת הפשטה לפרוטוקול ה-**MIDI**, אך עדיין מכילה** API **יחסית** "Low-Level" **שדורש היכרות עם שמות אירועים ב-**MIDI **בשביל אחזור ועדכון נתונים בקובץ. המחלקה** DryWetMidiAdapter **אחראית על התעסקות עם ה-"עבודה שחורה" הזו של אינטראקציה מול ה-**API **של הספרייה החיצונית, ולספק לקליינטים (יתר רכיבי הפיתוח בשכבה זו ובשכבות אחרות במערכת) מנשק** High-Level **פשוט, נקי וקל לתפעול.**

משתנים: **המחלקה מגדירה את סט המשתנים הפרטיים הבאים –**

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

בנאים: **המחלקה תומכת בשני בנאים – האחד מקבל כקלט נתיב לקובץ המידי והשני מקבל זרם בתים של הקובץ (שימושי בעיבוד תוכן קובץ שכבר נקרא ונמצא בזיכרון התכנית) –**

|  |  |
| --- | --- |
| Description | Signature |
| Construction based on given path to the actual MIDI file. | DryWetMidiAdapter(string) |
| 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. | DryWetMidiAdapter(Stream, bool) |

מתודות – **מעבר למאפיינים והמתודות במנשק** [IMidiFile](#_IMidiFile_Interface) **שממומשות במחלקה זו, מחלקה זו מגדירה גם שתי מתודות פרטיות ייעודיות** שאחראיות על **אדפטציה מול הספרייה החיצונית** [DryWetMidi](https://github.com/melanchall/drywetmidi) ע"י המרה בין הייצוג פנימי של ישויות המערכת אל ייצוגן הספרייה החיצונית ולהפך: מתודה אחת ([ConvertMelodyToTrackChunk](#ConvertMelodyToTrackChunk)) אחראית להמיר ייצוג של מנגינה המורכבת מישויות התאוריה המוסיקליות שתוארו בפרק [ייצוג ישויות מוסיקליות](#_ייצוג_ישויות_מוסיקליות)אל ישות חיצונית המייצגת רצועה בקובץ MIDI בספרייה החיצונית, ומתודה שנייה ([ConvertTrackChunkToMelody](#ConvertTrackChunkToMelody)) מבצעת את ההמרה בכיוון ההפוך. המרות אלו נדרשות לטובת שימוש בשירותי הספרייה החיצונית: אלגוריתמי ההלחנה עובדים על הישויות המוסיקליות, **ומפיקים בסיום תהליך ההלחנה מנגינה, שהיא פשוט אוסף של תיבות (ישויות** [IBar](#_IBar_Interface)**). בכדי לשבץ תיבות אלו בקובץ** MIDI **יש להמיר אותם לייצוג מתאים לספרייה החיצונית שמספקת את שירותי ה-**MIDI **השונים. באופן אנלוגי, לאחר קריאת קובץ** MIDI**, יש להמיר אותה מהייצוג בספרייה החיצונית כרצועה לסדרת תווים בתיבה ע"י הייצוג הפנימי במערכת. להלן תקציר המתודות –**

|  |  |
| --- | --- |
| Description | Method Name |
| 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](https://github.com/melanchall/drywetmidi)) representation. | ConvertMelodyToTrackChunk |
| Converts a midi track from a midi file represented in in external third-library ([DryWetMidi](https://github.com/melanchall/drywetmidi)) representation into a collection of musical notes in internal representation. | ConvertTrackChunkToMelody |

להלן דיאגרמת מחלקה עבור DryWetMidiAdapter –

****

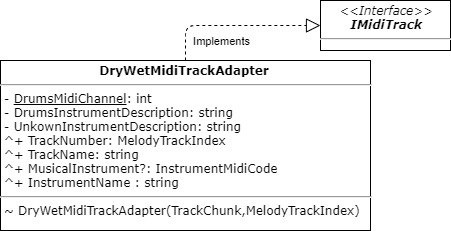
### מחלקת מעטפת לרצועת מידי (DryWetMidiTrackAdapter Class)

**מחלקה זו (**(DryWetMidiTrackAdapter **מממשת את המנשק** [IMidiTrack](#_IMidiTrack_Interface)**. בדומה למחלקה** [DryWetMidiAdapter](#_DryWetMidiAdapter_Class) **היא עושה זאת** **תוך שימוש בספריית עזר חיצונית –** [DryWetMidi](https://github.com/melanchall/drywetmidi), **שאותה היא עוטפת** **ומסתירה (מכמיסה), כך שכלפי חוץ נחשף רק המנשק הפשוט** [IMidiFile](#_IMidiFile_Interface) **שהקליינטים מצפים לו, ומכאן שם המחלקה (על שם דפוס העיצוב** Adapter**). יתר על כן, מאחר שזוהי אותי ספריית עזר חיצונית שמשמשת את המחלקה** [DryWetMidiAdapter](#_DryWetMidiAdapter_Class)**, ומאחר שהבנאי שלה תלוי בפרמטר מהספרייה החיצונית שאינו אמור להיות חשוף מחוץ למחלקה, המחלקה של ייצוג רצועה הוגדרה כמחלקה פנימית פרטית** (inner class) **המקוננת בתוך המחלקה המייצגת קובץ. זה גם מתיישב עם העובדה שרצועה רלוונטית רק תחת קונטקסט של איזשהו קובץ, כך שממילא קליינטים מחוץ למחלקה לא אמורים לקבל גישה ישירה לרצועה כישות עצמאית, אלא כישות המורכבת בתוך ישות של קובץ, כלומר הגישה לרצועות היא מתוך הקובץ המכיל אותן.**

**המחלקה אינה מגדירה פונקציונאליות נוספת מעבר לזו של המנשק** [IMidiTrack](#_IMidiTrack_Interface)**. כזכור, מנשק זה מספק רק מבנה (**POCO**) עם סט מאפיינים לקריאה בלבד, כך שכל מהות מחלקה זו היא לספק מימוש לאחזור מאפיינים אלו תוך שימוש ב-**API **של הספרייה החיצונית.**

בנאים: **המחלקה תומכת בבנאי יחיד המקבל שני פרמטרים – פרמטר ראשון הוא רצועת** MIDI **מטיפוס** [TrackChunk](https://melanchall.github.io/drywetmidi/api/Melanchall.DryWetMidi.Core.TrackChunk.html) **שמוגדר בספרייה החיצונית** [DryWetMidi](https://github.com/melanchall/drywetmidi)**, ופרמטר שני הוא מספרה הסידורי של הרצועה בתוך קובץ ה-**MIDI **שמכיל אותה. כאמור, הפרמטר הראשון הוא טיפוס מהספרייה החיצונית שאינו אמור להיות מוכר מחוץ למחלקה, אולם מאחר שזוהי מחלקה פרטית מקוננת, הגישה לבנאי נעשית רק מתוך המחלקה המכילה שממילא עושה שימוש בעצמה בספרייה החיצונית.**

**להלן דיאגרמת המחלקה –**

****

### מחלקת מפעל לישויות מידי (MidiFactory Class)

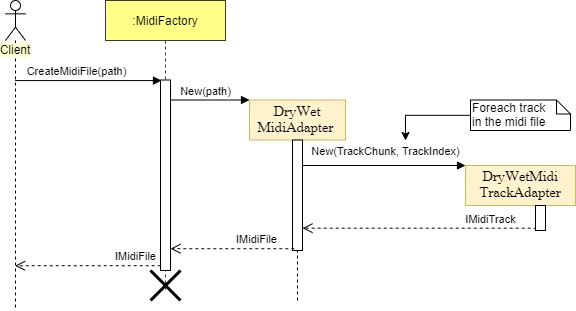
**מחלקה זו היא מחלקת שירות סטטית המספקת מתודות שירות ליצירת מופעים קונקרטיים שמממשים את המנשק** [IMidiFile](#_IMidiFile_Interface)**, ומכאן שם המחלקה (על שם דפוס העיצוב** FactoryMethod**). היא כוללת שני מתודות סטטיות ליצירת מופעים –**

internal static IMidiFile CreateMidiFile(string midiFilePath) {…}

internal static IMidiFile CreateMidiFile(Stream stream, bool disposeStream = false) {…}

**מוטיבציה מאחורי מחלקה זו היא לשבור את התלות של קליינטים במימוש ספציפי של שירותי ה-**MIDI**, ולהשאיר תלות רק באבסטרקציה, דהיינו רק במנשק עצמו. קליינטים המעוניינים בשימוש בשירותי** MIDI **יפנו למחלקת ה-**MidiFactory **בבקשה לקבלת מופע של המנשק, והמחלקה תהיה אחראית לספק מופע שכזה, וכל זאת בלי שהקליינטים צריכים להכיר את המחלקות שמספקות את המימוש של המנשק בפועל מאחורי הקלעים.**

**נכון לעכשיו, המימוש של ה-**Factory **כולל פשוט יצירת מופעים של מחלקת ה-**[DryWetMidiAdapter](#_DryWetMidiAdapter_Class)**, שכן זהו המימוש היחידי המוגדר כעת במערכת. אם בעתיד יהיה צורך בהחלפת המימוש של שירותי ה-**Midi**, מלבד המימוש עצמו של המנשק** [IMidiFile](#_IMidiFile_Interface) **(ו-**[IMidiTrack](#_IMidiTrack_Interface) **במידת הצורך) כל שיידרש הוא לבצע את ההחלפה במימוש הפנימי של מתודות ה-**Factory**, כלומר, לקליינטים עצמם ההחלפה תהיה "שקופה", הם ימשיכו לפנות כרגיל אל ה-**Factory **בבקשה לקבלת ספק שירותי** MIDI **ויקבלו מופע של המנשק שמספק שירות זה, מבלי בכלל לדעת מהי המחלקה הקונקרטית שמספקת את המימוש. להלן דיאגרמת רצף הממחישה כיצד לקוחות מבקשים ומקבלים מופע של** IMidiFile **שכבר עוטף בתוכו אוסף רצועות מטיפוס** IMidiTrack **–**

****

## הלחנת מנגינות (Composition Service)

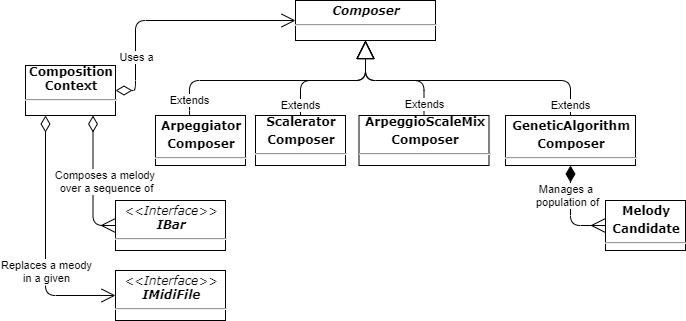
תת-שכבה זו של הלחנת מנגינות מחולקת למספר מרחבי שמות (Namespaces) כדלקמן –

|  |  |
| --- | --- |
| 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 –

|  |  |  |  |
| --- | --- | --- | --- |
| # | תפקיד | סוג רכיב פיתוח | שם הרכיב |
| 1 | מחלקת קונטקסט ההלחנה | Class | [CompositionContext](#_מחלקת_קונטקסט_ההלחנה) |
| 2 | מחלקת מלחין אבסטרקטי | Class | [Composer](#_מחלקת_מלחין_אבסטרקטי) |
| 3 | מחלקת מלחין פירוקי אקורדים | Class | [ArpeggiatorComposer](#_הלחנה_עם_אלגוריתמים) |
| 4 | מחלקת מלחין סולמות | Class | [ScaleratorComposer](#_הלחנה_עם_אלגוריתמים) |
| 5 | מחלקת מלחין סולמות ופירוקי אקורדים | Class | [ArpeggioScaleMixComposer](#_הלחנה_עם_אלגוריתמים) |
| 6 | מחלקת מלחין עם אלגוריתם גנטי | Class | [GeneticAlgorithmComposer](#_GeneticAlgorithmComposer_Class) |
| 7 | מחלקת מנגינה מועמדת באלגוריתם | Class | [MelodyCandidate](#_המחלקה_MelodyCandidate) |
| 8 |  | Class |  |
| 9 | אוסף קבועים לייצוג אלגוריתמי הלחנה | Enumeration | [CompositionStrategy](#_CompositionStrategy_Enumeration) |
| 10 | מפעל יצירת מלחין ע"פ אלגוריתם הלחנה | Class | [ComposerFactory](#_CompositorFactory_Class) |
| 11 | מחלקת Extension Methods לשירותי הלחנה | Class | [ComposerExtensions](#_הרחבות_(CompositorExtensions)) |
| 12 | מחלקת Extension Methods ל-Enums | Class | [EnumExtensions](#_EnumExtensions) |
| 13 | קבועי עזר שונים לסיווג פרמטרים המועברים למתודות השונות להלחנה. | Enumeration | [ChordNoteMappingSource](#_ChordNoteMappingSource) |
| 14 | Enumeration | [DurationSplitRatio](#_DurationSplitRatio) |
| 15 | Enumeration | [NoteSequenceMode](#_NoteSequenceMode) |
| 16 | Enumeration | [OverallNoteDurationFeel](#_OverallNoteDurationFeel) |
| 17 | Enumeration | [Permutation](#_Permutation) |
| 18 | Enumeration | [PitchRangeSource](#_PitchRangeSource) |
| 19 | Enumeration | [SortOrder](#_SortOrder) |

להלן דיאגרמת מחלקה המציגה את הישויות המרכזיות במודול זה והקשרים שביניהן –



### מחלקת קונטקסט ההלחנה (CompositionContext Class)

המחלקה CompositionContext משמשת כרכיב ה-context בדפוס העיצוב Strategy Pattern, בהקשר של קונטקסט לחיבור (הלחנה) של מנגינות, ומכאן שם המחלקה.

מחלקה זו אחראית על ניהול האינטראקציה ואינטגרציה עם כלל הרכיבים השונים בשכבת הלוגיקה העסקית, ונמצאת בקו החזית של השכבה מול הקליינטים: שכבות אחרות המעוניינות בשירותי ההלחנה פונים אך ורק למחלקת ה-CompositionContext והיא זו שאחראית על עיבוד הבקשה שלהם, בדיקת תקינות הקלט, ובמידה והבקשה תקינה, מחלקה זו אחראית להעביר אותה הלאה אל הגורמים הרלוונטיים להמשך טיפול – ראשית היא מעבירה את הבקשה אל אלגוריתם הלחנה לחיבור מנגינה, ולאחר מכן עם קבלת מנגינה מאלגוריתם ההלחנה המחלקה פונה לתת-שכבת שירותי ה-MIDI לשיבוץ המנגינה שהולחנה בקובץ MIDI שאותו לבסוף היא מחזירה כפלט לקליינטים.

באופן זה, הקליינטים לא צריכים להיות מעורים ב-low-level של פרטי המימוש של מה מבוצע ע"י אלגוריתם הלחנה ומה מבוצע ב-MIDI. הם יכולים פשוט לפנות למחלקת קונטקסט ההלחנה בבקשה למנגינה וזו תדאג לנהל את כל האופרציה מאחורי הקלעים ולספק להם את הפלט המבוקש תוך הסתרת כל הפרטים הטכניים. במילים אחרות, מחלקה זו משמשת גם כמעין "Facade" של שכבת הלוגיקה העסקית.

יתר על כן, שימוש בדפוס העיצוב Strategy Pattern עבור אלגוריתם ההלחנה מאפשר לקליינטים לבחור את אלגוריתם ההלחנה שבו ייעשה שימוש לחיבור המנגינה, וכל זאת בזמן ריצה ללא צורך בהידור מחדש של התכנית. נכון לעכשיו המימוש העיקרי מבין אלגוריתמי ההלחנה הוא אלגוריתם גנטי, שהינו אלגוריתם המבוסס על שיטות חיפוש היוריסטיות למציאת פתרונות אופטימאליים (במקרה הזה – מנגינות שנשמעות הכי "טוב"), יתר פירוט על אלגוריתם זה בהמשך בפרק [הלחנה עם אלגוריתם גנטי](#_הלחנה_עם_אלגוריתם). מאחר שזהו האלגוריתם העיקרי, הוא נבחר כברירת מחדל אם הקליינט לא ציין בחירה אחרת. מלבד אלגוריתם זה, הוגדרו מספר אלגוריתמי הלחנה בסיסיים ביותר שמחוללים רצף תווים על בסיס מהלך האקורדים, הסולמות הממופים לאקורדים, או שילוב ביניהם. אלו כאמור אלגוריתמים מאוד בסיסיים ולמעשה משמשים כאבני בניין באלגוריתם הגנטי המרכזי. הם הוגדרו כאלגוריתמים עצמאיים לבדיקה והמחשה של אפשרות השימוש בדפוס העיצוב של האסטרטגיה, כלומר לבדיקה והמחשה של היכולת לשנות את אלגוריתם ההלחנה המבוקש בזמן ריצה. ההמלצה היא בכל באופן לעשות שימוש באלגוריתם המרכזי – האלגוריתם הגנטי, ואכן, מערכת ה-Web חושפת כעת רק את האלגוריתם הגנטי, אך ניתן לשנות זאת בנקל הן ביישום ה-Web-י והן במערכת האב-טיפוס של מימוש יישום ה-PC ב-Windows Forms ו/או ביישום ה-Console.

#### בנאים (Constructors)

**המחלקה מגדירה שני בנאים, המקבלים את פרטי הקטע המוסיקלי שעבורו יש להלחין את המגינה החדשה. בנאי אחד מצפה לקבל מופעים "חיים" בזיכרון של תוכן קובץ ה-**MIDI **ומהלך 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 אתחול מופע הקונטקסט עם פרטי הקטע המוסיקלי תוך בדיקות תקינות נתונים של תוכן ה-**MIDI**, האקורדים, התאימות ביניהם והתאימות בין קובץ ה-**MIDI **לאינדקס הרצועה שמציינת את מספר רצועת המנגינה המקורית 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)

**להלן השדות והמאפיינים המוגדרים במחלקה –**

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

**כפי שניתן להבחין, מחלקה זו כוללת תכונות רבות, ובהתאם יש לה לא מעט תלויות (**Dependencies**) – תלות במנשק המייצג את קובץ ה-**MIDI**, תלות באלגוריתם ההלחנה (**Composition Strategy**) ועוד. חלק מתלויות אלו "מוזרקות" פנימה באמצעות הבנאי, דוגמת קובץ ה-**MIDI**, ואילו אחרות מוזרקות כפרמטרים למתודות, למשל אלגוריתם ההלחנה נשלח כארגומנט למתודת ההלחנה בפועל, כך שניתן יהיה להשתמש באותו מופע של** CompositionContext **על שיר מסוים ובכל בקשת** **הלחנה לספק אלגוריתם הלחנה אחר.**

#### מתודות המחלקה (Methods)

**המחלקה מגדירה מספר מתודות. ניתן לחלק אותן למספר קטגוריות:** בדיקת תקינות, קריאת קלט, מתודות עזר לתהליך הרכבת קובץ הפלט ומתודת ההלחנה עצמה –

* **מתודות בדיקות תקינות:** מתודות אלו מחזירות ערך בוליאני המחווה על תוצאת הבדיקה. להלן בדיקות התקינות המוגדרות במחלקה –

|  |  |
| --- | --- |
| Description | Method |
| 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. | AreBarsCompatible |
| Checks if the requested pitch range is valid. | IsPitchRangeValid |
| Checks if the mentioned index for identifying the existing melody track in the MIDI file is valid. | IsMelodyTrackIndexValid |

* **מתודות קריאת קלט:** מתודות אלו אחראיות על קריאת התכנים מקבצי הקלט של ה-MIDI והאקורדים (ו/או מזרם של בתים במקום מתוך קובץ) והחזרת "ידיות" המפנות למופעים של תכנים אלו – במקרה של תוכן MIDI זהו פשוט מופע של המנשק IMidiFile, ובמקרה של תוכן רצף אקורדים הידית היא הפנייה לאוסף של תיבות המכילות את האקורדים כפי שמתואר באופן טקסטואלי בקובץ הקלט / זרם הבתים מהקלט.מאחר שפעילות של קריאת קלט אינה תלויה במופע ספציפי של מחלקת קונטקסט ההלחנה, מתודות אלו הוגדרו כמתודות סטטיות. להלן שמות המתודות ותיאור תמציתי שלהן –

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

**פורמט קובץ האקורדים**: להבדיל מפורמט קובץ MIDI שהינו פורמט המוגדר היטב הודות לפרוטוקול MIDI שהפך לסטנדרט, אין פורמט סטנדרטי אחיד לקובץ המכיל תיאור של אקורדים. לפיכך, בפרויקט הוגדר פורמט ייעודי לקלט הטקסטואלי המתאר מהלך אקורדים. פורמט זה מתואר ב[נספח בסוף תת–סעיף זה](#ChordProgressionFormat).

* **מתודות בדיקות תקינות:** מתודות אלו מחזירות ערך בוליאני המחווה על תוצאת הבדיקה. להלן בדיקות התקינות המוגדרות במחלקה –

|  |  |
| --- | --- |
| Description | Method |
| 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. | AreBarsCompatible |
| Checks if the requested pitch range is valid. | IsPitchRangeValid |
| Checks if the mentioned index for identifying the existing melody track in the MIDI file is valid. | IsMelodyTrackIndexValid |

* **מתודות עזר:** המחלקה מגדירה שתי מתודות סטטיות האורזות 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 –

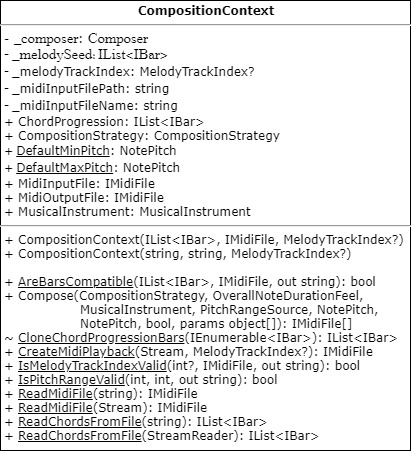
|  |  |
| --- | --- |
| Description | Method |
| Duplicates a given midi file, removes its melody track, and returns the result (midi playback). | CreateMidiPlayback |
| Duplicates a given chord progression and removes any empties out any note sequences from the containing bar sequence. | CloneChordProgressionBars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

* **מתודת ההלחנה:** זוהי המתודה שמספקת את שירות ההלחנה, כלומר זוהי המתודה הראשית שהקליינטים פונים אליה במטרה לקבל מנגינה חדשה –

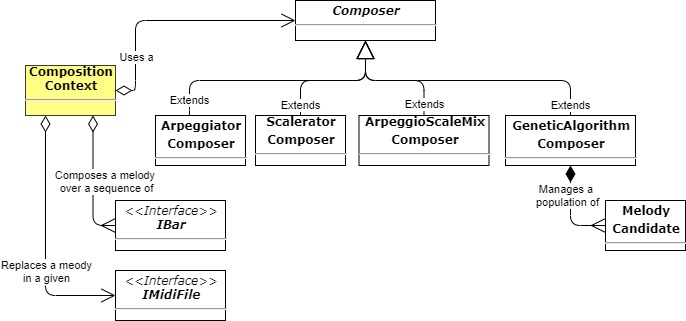
|  |  |
| --- | --- |
| Description | Method |
| Composes a solo-melody over this composition's midi playback file and chord progression, using the additional user preferences and constraints parameters. | Compose |

#### דיאגרמת מחלקה (Class Diagram)

להלן דיאגרמת מחלקה של מחלקת קונטקסט ההלחנה –



**להלן דיאגרמה נוספת המתמקדת בקשר שבין מחלקת קונטקסט ליתר הרכיבים המפורטים בהמשך השייכים לתת-שכבה זו של שירות הלחנת המנגינות –**

****

#### 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 רווח) 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 רבעים לתיבה (המונה הוא 3 והמכנה 4).
    - לאחר משקל התיבה תירשם סדרת האקורדים בתיבה מופרדים ביניהם ברווח 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 לאורך שתי פעימות.
    - שמות תווי שורש האקורד המותרים לשימוש הם השמות המקובלים בטרמינולוגיה של המוסיקה המערבית: A עבור לה, B עבור סי, C עבור דו, וכן הלאה עד ל-G עבור סול. בכדי לציין תו עם סימן התק (דיאז או במול), יש להוסיף לתו את הסיומת Sharp או Flat בהתאמה. למשל CSharp מציין את התו דו-דיאז C#, ואילו BFlat למשל מציין את התו סי-במול Bb.
    - שמות סוגי האקורדים המותרים הם אלו שבעמודה השמאלית בטבלה שלהלן (עמודת 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 מוקטנת (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 (1-4-5) | MajorSuspended4 |
| אקורד מז'ור משולש עם דרגה חמישית מוגדלת (1-3-5#) | 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:



### מחלקת מלחין אבסטרקטי (Composer Abstract Class)

המחלקה Composer היא מחלקה אבסטרקטית המשמשת כרכיב ה-strategy בדפוס העיצוב Strategy Pattern, בהקשר של אסטרטגיה אלגוריתמית להלחנת מנגינות. המחלקה אחראית על הגדרת מנשק פונקציונאלי משותף לכל המחלקות הקונקרטיות המממשות אלגוריתם הלחנה, וכן לספק תשתית עזר של שירותי הלחנה גנריים בסיסיים שאינם תלויים באלגוריתם הלחנה ספציפי, אלא יהיו שימושיים לכלל המחלקות היורשות ויוכלו להוות אבני-בניין בבניית שגרות מורכבות ופרטניות יותר. שירותים כאלה יכולים לכלול למשל היפוך או ערבול סדר של רצף תווים נתון, יצירת רצף אקראי של משכי שהייה, עדכון גובה צליל בכמה חצאי-טונים, הארכה/קיצור משך שהייה של צליל וכד'.

המחלקה עצמה היא כאמור אבסטרקטית. זה בא לידי ביטוי בכך שהיא אינה מספקת אלגוריתם הלחנה בעצמה, אלא מגדירה מתודה אבסטרקטית – GenerateMelody שעל המחלקות היורשות ממנה לממש. כל מחלקה קונקרטית שיורשת מ-Composer מייצגת איזשהו אלגוריתם הלחנה שמבצע את ההלחנה במסגרת מימוש המתודה האבסטרקטית GenerateMelody. כרגע המערכת מספקת מימושים של ארבעה אלגוריתמי הלחנה: שלושה מהם אלגוריתמים מאוד בסיסיים שפשוט מחוללים רצף תווים המבוסס בהתאמה על האקורדים, הסולמות הממופים לאקורדים, וצירוף של השניים, ואילו המימוש הרביעי, שהוא ה-"מח" ו-"לב" המערכת – הוא מימוש אלגוריתם גנטי, המפורט בהמשך.

#### תכונות המחלקה (Fields, Properties)

**להלן השדות והמאפיינים המוגדרים במחלקה –**

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

#### מתודות המחלקה (Methods)

**מתודות עבור שירותי הלחנה בסיסיים:** מתודות אלו משמשות כאבני בניין בסיסיות לביצוע מניפולציות ועדכונים שונים על רצפי תווים, משכי שהייה, אקורדים ותיבות. אבני בניין אלו יכולים להיות לעזר בכל פעילות הלחנה ללא תלות באופן המימוש של אלגוריתם ההלחנה, ולכן הן מרוכזות במחלקת האב האבסטרקטית כך שכל מחלקה יורשת תוכל לבחור להשתמש בהם ע"פ הצרכים שלה. כלל המתודות הללו מוגדרות כ-private protected על מנת שיהיו זמינות אך ורק בגוף המחלקה עצמה ולמחלקות היורשות, וכן הן מוגדרות כ-virtual על מנת שמחלקות בנות המעוניינות בכך יוכלו לדרוס (override) את פונקציונאליות ברירת המחדל ולהתאים אותה לצרכיהן. חלק ממתודות השירות שלהלן הם בעצמן גנריות ומשמשות כאבני בניין להגדרת מתודות שירות ספציפיות יותר עם וריאציות שונות. להלן פירוט כלל מתודות השירות–

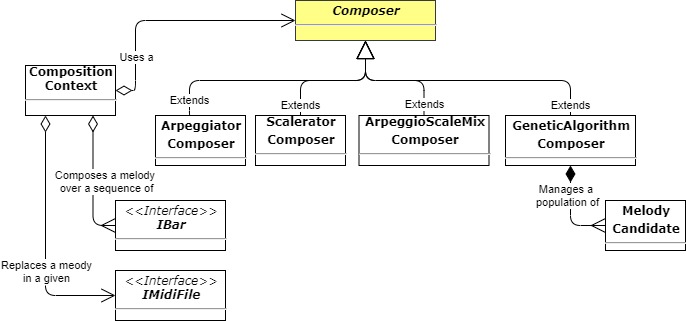
|  |  |
| --- | --- |
| Description | Method |
| General arpeggiator initializer that initializes an entire note sequence for a given sequence of bars, based on the bars chord progression arpeggio notes and requested note sequence mode (ascending, descending or zigzag). | ArpeggiatorInitializer |
| Initializes 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. | ArpeggiatorInitializerAscending |
| ArpeggiatorInitializerDescending |
| ArpeggiatorInitializerChordZigzag |
| ArpeggiatorInitializerBarZigzag |
| Selects a random note in a bar and changes its pitch to another note from the requested mapping source (chord/scale). | ChangePitchForARandomNote |
| 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. | NoteDurationSplit |
| 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. | NoteSequenceInitializer |
| Generates a permutation of the bar's note sequence & replaces the original sequence with the permutated sequence (shuffled/reversed, etc). | PermutateNotes |
| Initializes a note sequence for a given sequence of bars, based both on the bars chord progression mapped scales & the arpeggio notes. | ScaleArpeggioeMixInitializer |
| General scalerator 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). | ScaleratorInitializer |
| 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. | ScaleratorInitializerAscending |
| ScaleratorInitializerDescending |
| ScaleratorInitializerChordZigzag |
| ScaleratorInitializerBarZigzag |
| Syncope's a bar's first note by connecting it to its preceding note (last note from preceding bar). | SyncopizeANote |
| Replaces a random hold note with a concrete note pitch. | ToggleAHoldNote |

מתודות אתחול והלחנה: **המתודות שלהלן נמצאות בשימוש ע"י כל המחלקות היורשות: הראשונה (**[InitializeCompositionParams](#InitializeCompositionParams)**) אחראית על אתחול שמבוצע ממש לפני פעילות ההלחנה. מתודת אתחול זו מוגדרת כ-**virtual **לצורך מתן האפשרות למחלקות יורשות לדרוס אותה (**override**) ולממש התאמות ע"פ צרכים שלהם. המתודה השנייה (**[*GenerateMelody*](#GenerateMelody)**) היא המתודה האבסטרקטית שמיועדת להכיל את הלוגיקה העסקית למימוש אלגוריתם ההלחנה, והמתודה השלישית היא שזו שהקליינטים מחוץ למחלקה מכירים. היא עוטפת את מתודת האתחול ומתודת ההלחנה ה-"אמיתית" (**GenerateMelody) שאחראית על ביצוע ההלחנה בפועל. להלן פירוט נוסף –

|  |  |
| --- | --- |
| Description | Method |
| Initialize general parameters such as duration default values and bounds, pitch range, melody seed, chord progression etc.  This methods 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. | InitializeCompositionParams |
| 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](#InitializeCompositionParams) method is done. | *GenerateMelody* |
| 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](#InitializeCompositionParams) for the initialization phase, and then executing the actual [*composition method*](#GenerateMelody) for generating the melody. | Compose |

#### דיאגרמות מחלקה

להלן דיאגרמות מחלקה המציגות את הקשר בין מחלקת המלחין האבסטרקטי ליתר הישויות בתת-שכבה זו, וכן את כלל המאפיינים והמתודות המוגדרות במחלקה –



דיאגרמת מחלקה המציגה את כלל המאפיינים והמתודות המוגדרות במחלקת הקונטקסט –

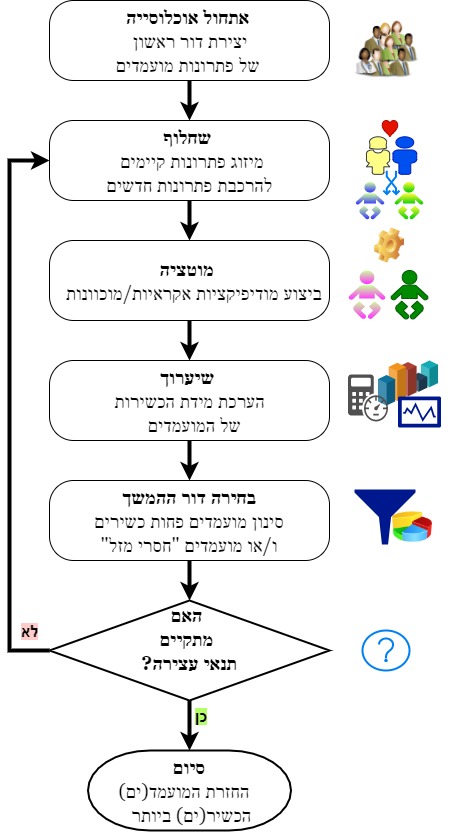
****

### הלחנה עם אלגוריתם גנטי

#### רקע לאלגוריתמים גנטיים

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

תיאור שלבי אלגוריתםגנטי טיפוסי**: האלגוריתם מתניע את תהליך האבולוציה המלאכותי עם אתחול דור ראשון של פתרונות מועמדים, שניתן לחולל הן באופן אקראי, והן באופן יזום על-סמך פרמטרים נבחרים. על בסיס אוכלוסייה זו של הדור הראשון מתחיל תהליך מחזורי (מעגלי) המדמה תהליכים אבולוציוניים מהביולוגיה (ראה איור משמאל):**

1. **בשלב ראשון מדמים תהליך של זיווג/רבייה ע"י שימוש באופרטור השחלוף שממזג חלקים משני פתרונות או יותר (ההורים) ובונה מהם פתרונות חדשים (צאצאים).**
2. **בשלב שני מדמים "רעש" שקיים בטבע כגון מוטציות גנטיות והשפעות סביבתיות הכופות שינוי הסתגלותי (אדפטיבי) על המועמד. שלב זה ממומש ע"י אופרטור מוטציה שאחראי לבצע מודיפיקציות תואמות (שינויים אקראיים ו/או שינויי הסתגלות) בחלק מהמועמדים.**
3. **בשלב שלישי מבצעים תהליך שיערוך (**fitness evaluation**) של אוכלוסיית הפתרונות המעודכנת בכדי לאמוד את טיב האיכות שלהם ע"פ הסיווג המתקבל מפונקציית הכשירות.**
4. **בשלב רביעי מדמים את הברירה הטבעית (**natural selection) **מהתאוריה של דארווין ע"י תהליך סלקציה שבורר מבין כל המועמדים באוכלוסייה הנוכחית אילו ימשיכו לדור (מחזור) הבא ואילו יישארו מאחור ו-"ייכחדו". קונספט הברירה הטבעית בא כאן לידי ביטוי ע"י מתן העדפה בבחירה למועמדים הכשירים ביותר, בהתאם להערכה שבוצעה בשלב הקודם.**
5. **לאחר שלב הסינון מסתיים מחזור והאלגוריתם מגיע לנקודת צומת דרכים שבה הוא צריך לקבל החלטה אם להמשיך בחיפוש ע"י התחלת מחזור נוסף בניסיון לשפר עוד יותר את האוכלוסייה הקיימת או שמא הגיעה העת לעצור (למשל אם הפתרונות שנמצאו טובים מספיק, או שמא בשל אילוצי זמן/כוח חישובי יש להסתפק בפתרונות הקיימים).**

להלן פסידו-קוד של האלגוריתם שתואר לעיל –

1. אתחל דור ראשון
2. בצע –
   1. שחלוף
   2. מוטציה
   3. שיערוך
   4. בחירת דור המשך
3. בדוק האם מתקיים תנאי העצירה –
   1. אם כן, החזר כפלט את המועמד(ים) הכשיר(ים) ביותר וסיים;
   2. אחרת, חזור לשלב 2.

בהקשר של המערכת הנידונה, בעיית האופטימיזציה הנתונה היא חיבור מנגינה, אוכלוסיית הפתרונות המועמדים היא קבוצת מנגינות שמתוכן על האלגוריתם למצוא את זו "הטובה" ביותר, דהיינו זו שנשמעת הכי "טוב". מה זאת מנגינה טובה? זו שאלה די סובייקטיבית, אולם ניתן להגדיר מדדים כלליים שונים ולתת למשתמש (המאזין) את היכולת לקבוע יחס סדר בין מדדים אלו, כך שהאלגוריתם יעדיף מנגינות שמצטיינות במדדים שהמשתמש מייחס להם חשיבות גבוהה יותר. זאת, יחד עם מתן האפשרות למשתמש לציין העדפות נוספות כגון מידת דחיסות הצלילים ומנעד נותנות איזשהו קירוב למה שהמשתמש יכול להחשיב כמנגינה טובה בעיניו (או ליתר דיוק, באוזניו).

פירוט נוסף על כל אחד משלבי האלגוריתם שלעיל, על אלגוריתמים גנטיים בכלל ודיון ביישום שלהם לחיבור מנגינות, ניתן למצוא [במסמך המצורף של עבודת סמינר בנושא](https://github.com/cwelt/Soloist/blob/master/WebApplication/App_Data/Seminar/GA_Seminar.pdf) –



#### תיאור רכיבי הפיתוח של האלגוריתם הגנטי

**חלוקת מחלקת האלגוריתם הגנטי למודולים**

כל אחד מהשלבים של האלגוריתם הגנטי שתוארו לעיל (אתחול, שחלוף, מוטציה, וכו') מכילים לא מעט לוגיקה. בכדי לשמור על סדר במחלקת האלגוריתם הגנטי (GeneticAlgorithmClass), המחלקה פוצלה סביב מספר קבצים (כל קובץ מוגדר כ-partial class), כאשר כל קובץ אחראי לטפל באחד המודולים של האלגוריתם. כל מודול מגדיר מתודת ניהול ראשית שנקראת מתוך האלגוריתם הגנטי המרכזי, ומתודות עזר פנימיות שמטפלות בפרטים הטכניים. להלן תקציר תוכן הקבצים (מודולים) השונים שמגדירים יחדיו את המחלקה GeneticAlgorithmClass, עם תקציר התוכן שלהם שם מתודת הניהול שלהם (המתודה שנקרית מתוך האלגוריתם המרכזי ) –

|  |  |  |
| --- | --- | --- |
| תוכן | מתודת ניהול מרכזית | שם קובץ (מודול) |
| הגדרת תזרים האלגוריתם הגנטי. | GenerateMelody | GeneticAlgorithmComposer.cs |
| אתחול דור ראשוני של מנגינות מועמדות. | PopulateFirstGeneration | Initialization.cs |
| שלב השחלוף של האלגוריתם. | Crossover | Crossovers.cs |
| שלב המוטציה של האלגוריתם. | Mutate | Mutators.cs |
| שלב הערכת טיב איכות המנגינות המועמדות. | EvaluateFitness | FitnessEvaluators.cs |
| שלב בחירת המנגינות שימשיכו לדור הבא. | SelectNextGeneration | Selectors.cs |

בנוסף למחלקת האלגוריתם הגנטי (GeneticAlgorithmClass), הוגדרה מחלקה נפרדת לייצוג מנגינה אינדיווידואלית המשתתפת כמנגינה מועמדת באלגוריתם הגנטי: MelodyCandidate**.**

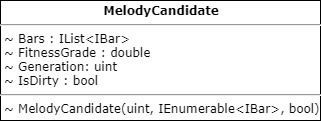
העמודים הבאים מכילים תיעוד מפורט על כל אחת ממחלקות אלו והמודולים השונים.

#### **MelodyCandidate Class**

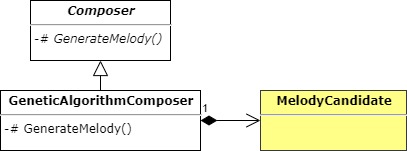
מחלקה זו מייצגת פתרון מועמד של מנגינה באלגוריתם הגנטי. מלבד המנגינה עצמה (רצף תיבות המכיל את אוסף הצלילים המרכיבים את המגינה), כל פתרון מועמד (דהיינו כל מופע של (MelodyCandidate מכיל גם מאפיינים הרלוונטיים לאלגוריתם הגנטי כגון הדור שאליו שייך המועמד (דהיינו מספר האיטרציה באלגוריתם הגנטי שבה המועמד נוצר) וציון מד האיכות שלו.

הפתרונות המועמדים השונים נוצרים ע"י מופע כלשהו של אלגוריתם גנטי, ומחזור החיים שלהם מוכל ותלוי במחזור החיים של האלגוריתם הגנטי שיצר אותם.

**להלן דיאגרמת מחלקה המציגה את המאפיינים והשיטות של המחלקה** MelodyCandidate **–**

****

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

****

מאפיינים**: להלן תיאור מאפייני המחל'קה** (Properties) **מדיאגרמת המחלקה שלעיל –**

|  |  |  |
| --- | --- | --- |
| Description | Type | Member Name |
| List of bars which contain the melody of this candidate. | IList<IBar> | Bars |
| The current score of this candidate, which signifies how good it is. | double | FitnessGrade |
| The generation that this candidate belongs in, i.e., what generation was he created at. | uint | Generation |
| 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. | bool | IsDirty |

בנאי**: המחלקה מגדירה בנאי יחיד המאתחל מועמד בהינתן לו מספר הדור (איטרציה נוכחית של האלגוריתם) שאליו הוא משתייך, המבנה ההרמוני שעליו יש לייצר מנגינה (רצף אקורדים המוכל ברצף התיבות הניתן כקלט) ופרמטר הקובע האם לאתחל 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 שאומדת מנגינה זו.

#### GeneticAlgorithmComposer Class

המחלקה GeneticAlgorithmComposer יורשת מהמחלקה האבסטרקטית Composer מממשת את אסטרטגיית ההלחנה שלה בגוף המתודה האבסטרקטית GenerateMelody תוך מימוש [תבנית הפסידו-קוד הכללית של אלגוריתמים גנטיים](#GenerateAlgoruthmPseudoCode) שהוצגה לעיל. להלן סגמנט מהקוד של מימוש 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;

}

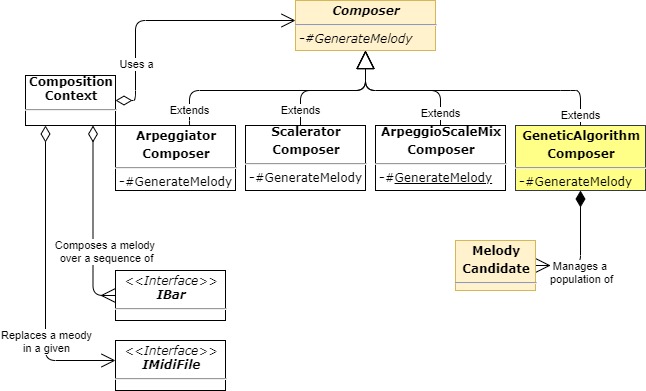
return …

}

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

מעבר למתודה GeneticAlgorithmComposer, במודול זה מוגדרת רק עוד מתודה אחת יחידה: InitializeCompositionParams, המתקבלת בירושה ממחלקת המלחין האבסטרקטי Composer ואחראית על אתחול ראשוני של פרמטרים לאלגוריתם ההלחנה. האלגוריתם הגנטי מנצל מתודה זו לאיפוס מספר הדור הגלובאלי לקראת מחזור ריצה חדש, אתחול אוכלוסיית המנגינות המועמדות ואתחול משקלים יחסיים לפונקציות השערוך (פירוט נוסף על משקלים אלו בפרק המתאר את [מודול השערוך](#_מודול_שערוך_(FitnessEvaluators.cs))).

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



להלן דיאגרמת מחלקה (class diagram) עם כלל המאפיינים והמתודות במחלקה זו –



#### תכונות המחלקה

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

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

#### 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 אופציונאלי, האתחול יכול לבסס את האתחול על סמך מנגינת בסיס כלשהי (initialization seed), למשל – המנגינה המקורית, או אתחול מ-"scratch" שאיננו מבוסס כלל על מנגינה קודמת אלא אך ורק על המהלך ההרמוני (אקורדים).

**מתודות אתחול**: האתחול מבוסס על אבני בניין – מתודות אתחול המוגדרות במחלקת האב [Composer](#_מחלקת_מלחין_אבסטרקטי). מתודות אתחול אלו מחוללות רצפי תווים על סמך האקורדים של המנגינה. השוני ביניהן הוא במקור המיפוי של התווים (האם מתוך צלילי האקורד או סולם שממופה אל מול האקורד), ובסדר התווים שברצף (האם הרצף בכל תיבה צריך להיות בעלייה, ירידה או משתנה לסירוגין).

**מתודת רישום**: המתודה RegisterInitializers אחראית על רישום מתודות האתחול השונות (אבני הבניין). היא נקרית מתוך בנאי המחלקה ורושמת את מתודות האתחול ברשימה ייעודית המיוצגת ע"י המאפיין \_initializers.

**מתודת ניהול האתחול**: המתודה PopulateFirstGeneration אחראית על ניהול כלל האופורציה סביב האתחול: קריאת מתודות האתחול מתוך הרשימה הייעודית, הפעלתן זו אחר זו בכדי להפיק מנגינות שונות, ביצוע מניפולציות ועיבוד נוסף בכדי לייצר וריאציות נוספות ובמידת הצורך לוקחת בחשבון גם מנגינת בסיס כגרעין (seed) לייחוס. מתודה זו היא המתודה הנקראת מתוך האלגוריתם הגנטי (דהיינו מתוך המתודה GenerateMelody) לטובת ביצוע האתחול. בסיום פעולתה המאפיין \_candidates המכיל את אוכלוסיית המנגינות המועמדות השונות מאוכלס ומלא.

**מוסכמות חתימה**: מתודות האתחול הן בעלות המוסכמה הבאה של חתימה אחידה –

(IEnumerable<IBar> bars(

כל מתודת אתחול מקבלת רצף תיבות ופועלת במקום (inplace) על רצף תיבות זה ללא החזרת פלט. המוסכמה על אחידות בחתימה מספקת גנריות ו-*דינאמיות*: מתודת ניהול האתחול אינה צריכה להכיר את המתודות האתחול הקונקרטיות השונות כל עוד הן נרשמות ברשימת מתודות האתחול \_initializers ע"י מתודת הרישום RegisterInitializers. במידה ויהיה צורך לעשות שימוש במתודת אתחול בעלת חתימה אחרת, רצוי יהיה להקים רשימה נוספת של delegates עם חתימות תואמות, ולעבוד מול אותה רשימה באופן גנרי בשלב הרישום בבנאי ואחזור במתודת הניהול ולא לפנות למתודת אתחול כזו או אחרת באופן מפורש.

**תקציר המתודות במודול:** להלן המתודות המוגדרות במודול זה (מתודות אתחול של אבני הבניין מוגדרות כאמור מחוץ למודול במחלקת האב Composer) –

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

**תיעוד מפורט ומלא של המתודות השונות נמצא בגוף קבצי קוד המקור.**

#### מודול שחלוף (crossover.cs)

**רקע**: מודול זה מרכז את פעילות השחלוף בקובץ ייעודי: crossovers.cs. המודול אחראי על מימוש אופרטור השחלוף לאלגוריתם הגנטי. אופרטור זה מצליב וממזג זוגות פתרונות (מנגינות) ומפיק מהם פתרונות חדשים – צאצאים, המורכבים מערבוב רכיבים של הפתרונות שיצרו אותם – "ההורים".

**מתודות השחלוף**: המתודה Crossover היא המתודה שנקראת ישירות מתוך האלגוריתם הגנטי עצמו (דהיינו ישירות מתוך המתודה ([GenerateMelody](#GenerateMelody). מתודה זו קוראת בתורה למתודה אחרת NPointCrossover המממשת שחלוף בשיטה הפופולרית n-point-crossover המשלבת חלקי פתרונות מהמועמדים ההורים לסירוגין ע"פ n נקודות הצלבה, כאשר n ניתן כפרמטר. בהקשר של מיזוג בין מנגינות, הנקודות מייצגות תיבות. אופן בחירת n הנקודות ההצלבה ממומש במתודות עזר, האחת בוחרת את הנקודות אקראית, והשנייה מבצעת בחירה אופטימאלית, במובן שנקודות הפיצול נבחרות באופן שמבטיח מעבר "חלק" במידת הניתן בין תווים שכנים בנקודות הפיצול, וע"י כך מניעת מצב שלאחר הפיצול תיווצר מנגינה עם קפיצות חריגות בין צלילים שכנים כתוצאה מחיבור חלקי מנגינות "לא קשורות" בגבהים שונים. להלן תקציר המתודות במודול זה, פירוט ותיעוד מלא נמצא בגוף קבצי קוד המקור –

|  |  |
| --- | --- |
| Description | Method |
| 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. | Crossover |
| Implements a crossover between two or more candidate participants in N distinct points. | NPointCrossover |
| Utility method for selecting crossover points in a way that minimizes the interval outcome the transition point after the crossover. | SelectOptimizedCrossoverPoints |
| Utility method for selecting n distinct crossover points randomly. | SelectRandomCrossoverPoints |

הערה: במימוש הקלאסי של שיטת הצלבה זו, המיזוג מבוצע בין זוג פתרונות (שני פתרונות בלבד), אולם המימוש הנוכחי מכליל את המימוש הקלאסי ומאפשר מיזוג אוסף *כלשהו* של פתרונות.

#### מודול מוטציה (mutators.cs)

**רקע**: סעיף זה מתאר את המודול האחראי על ביצוע מוטציות (שינויים) למנגינות. מודול זה מרוכז בקובץ: mutators.cs. המוטציות מדמות שינויים הסתגלותיים אבולוציוניים ו-"רעשים" הקיימים בטבע, ומעבירים את המנגינות השונות תחת סדרות של עדכונים (לטוב או לרע). המוטציות בעצם מהוות צעדים במרחב החיפוש. כל שינוי לוקח את החיפוש לכיוון מסוים, והשאיפה היא שלאורך זמן, תוך שיתוף פעולה עם תהליכי השערוך והסלקציה, שדואגים לסנן 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/תיבה שלמה (למשל ערבוב סדר התווים בחצי התיבה הראשונה). חלק מהשינויים הם ניטרליים, במובן שהם פועלים על המנגינות השונות באופן אקראי ללא מודעות אם השינוי ישפר או ירע את דירוג טיב האיכות, וחלק מהשינויים הם מוכוונים יותר ומוטים בכיוון מסוים בכדי לשפר איזושהי מגרעה ספציפית שאותרה.

**סקלביליות:** גישה פופולרית באלגוריתמים גנטיים היא להתחיל את החיפוש אחר הפתרונות האופטימאליים בחיפוש לרוחב, ובהמשך לעבור לחיפוש לעומק, דהיינו חיפוש שמכסה שטח רחב היקף אך סורק אותו רק באופן שטחי, ובהמשך לאחר קצת לימוד של השטח והסקת השערות לגבי אזורים שנראים קצת יותר אטרקטיביים, לצמצם את החיפוש לאותם אזורים אך לבצע בהם חיפוש יסודי. בכדי לאפשר לאלגוריתם להתאים את היקף החיפוש באופן דינאמי בזמן ריצה, לכל מוטציה מוקצה ערך ממשי בין 0 ל-1 שמייצג את ההסתברות לביצוע השינוי על מנגינה נתונה באיטרציה מסוימת של האלגוריתם. בנוסף מוגדרים פרמטרים שמשמשים לשינוי הסתברות זו בכל איטרציות עם התקדמות האלגוריתם. בכדי לתמוך הגישה הפופולרית שלעיל, ההסתברויות מופחתת בכל איטרציה מה שמביא לאורך זמן לפחות ופחות שינויים והתכנסות/התייצבות לקראת עצירה. המוטציות וההסתברויות המוקצות להן באתחול מוגדרות בטבלת Dictionaryבמאפיין barMutations\_, כאשר המפתח הוא delegate (Action) למתודה האחראית על ביצוע המוטציה, והערך הוא ההסתברות של המוטציה להתבצע. הטבלה נבנית במתודה RegisterMutators שמופעלת מתוך הבנאי. מדרגת השינוי הנוכחית היא קבועה לכלל המוטציות והיא מוגדרת במשתנה הקבוע MutationProbabilityStep.

**אופן ביצוע המוטציות**: כל מוטציה מוגדרת במתודה ייעודית האחראית על ביצוע שינוי ספציפי. מתודת ניהול המוטציות – Mutate, שנקראת מתוך האלגוריתם הגנטי, אחראית על תפעול וביצוע המוטציות השונות. ראשית היא בוחרת את האוכלוסייה שעליה היא תבצע מוטציות: בכדי לשמר איזושהי יציבות, לא כל המנגינות עוברות שינוי בכל איטרציה, אלא רק אוכלוסייה חלקית, חלקה מורכבת ממנגינות חדשות שנוצרו בשלב השחלוף באיטרציה הנוכחית, וחלקה ממנגינות "וותיקות" שנוצרו באיטרציות קודמות. לאחר בחירת האוכלוסייה, כל מועמד שנבחר עובר סדרה של שינויים אקראיים: גם המוטציות נבחרות באקראיות, וגם המוטציות שנבחרות לא בהכרח יבוצעו, זה תלוי בהסתברות שלהם לביצוע. לאחר ביצוע סדרת השינויים המנגינה המועמדת מסומנת בדגל IsDirty, בכדי שהאלגוריתם "יידע" שיש להעריך את טיב האיכות שלה מחדש לאור השינוי (מנגינות 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 סקלביליות.

**מוסכמות חתימה**: מתודות המוטציה הן בעלות המוסכמה הגנרית הבאה בחתימה שלהן –

([MelodyCandidate](#_המחלקה_MelodyCandidate) melody, int? barIndex = null(

הגנריות של מוסכמה זו מספקת *דינאמיות*: מתודת ניהול המוטציות אינה צריכה להכיר את המתודות הקונקרטיות כל עוד הן נרשמות בטבלת ה-Dictionary של המוטציות barMutations\_ ע"י מתודת האתחול RegisterMutators. לפיכך, אם יש צורך במוטציה בעל חתימה אחרת, כדי לשמר את הגנריות נדרש לתחזק טבלה נוספת של delegates עם חתימה תואמת, לרשום בה את המתודות הרלוונטיות באתחול ולגשת לטבלה זו במתודת הניהול, בנוסף לזו הקיימת. החיסרון באחידות הוא שמתודות המוטציה הקיימות צריכות להתיישר: לפיכך אם יש מתודת שינוי שכן פועלת על תיבה ספציפית, ואינדקס מספר התיבה לא סופק לה, עליה להמציא מספר תיבה בעצמה (למשל ע"י הגרלת מספר אקראי).

**מוסכמות שם**: מתודות המוטציה הן בעלות מוסכמת השם הבאה: <Description>Mutation, כאשר <Description> מוחלף בתיאור השינוי (שם עצם/פועל) למשל: ReverseBarNotesMutation.

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

|  |  |
| --- | --- |
| Description | Method |
| Selects a random note in the requested bar and changes its pitch to one of the chords pitches. | ChordPitchMutation |
| 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. | DurationUnifyMutation |
| Replaces a random note in the given bar with two new shorter notes with durations that sum up together to the originals note duration.  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.  Regarding the new durations, they are set according to the requested [ratio](#_DurationSplitRatio) or a randomly selected ratio. | DurationAnticipationSplitMutation |
| DurationDelaySplitMutation |
| DurationEqualSplitMutation |
| 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 |
| ReverseChordNotesMutation |
| Selects a random note in the requested bar and changes its pitch to one of the scale pitches. | ScalePitchMutation |
| Swaps the positions of two chord notes in the given bar. | SwapTwoNotesMutation |
| Syncope's 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). | SyncopedNoteMutation |
| Replaces a random hold note with a concrete note pitch and vice-versa. | ToggleFromHoldNoteMutation |
| ToggleToHoldNoteMutation |

חלק גדול מהמוטציות מתבסס על אבני הבניין שכבר הוגדרו במחלקת האב Composer.

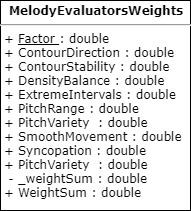
#### מודול שערוך (FitnessEvaluators.cs)

**רקע**: סעיף זה מתאר את המודול האחראי על שערוך טיב איכות המנגינות. מודול זה מרוכז בקובץ: FitnessEvaluators.cs. המודול מגדיר ומנהל מתודות האחראיות על שערוך טיב איכות המנגינות השונות שהאלגוריתם מפיק. שיערוך זה משמש כשלב הכנה מקדים לתהליך הסלקציה של האלגוריתם, שקובע אלו מבין כל המנגינות ימשיכו לשלב הבא, תוך מתן שיקול משמעותי לדירוג הניתן להם ע"י שגרות השערוך.

**אופן ביצוע השערוך**: עבור כל מדד המשמש להערכת מנגינה נתונה, מגדירים מתודת שערוך ייעודית ומשקל יחסי. מתודת השערוך הייעודית אחראית לאמוד את המנגינה ביחס למדד הנתון ולהחזיר מספר ממשי בין 0 ל-1, המייצג ציון שנמצא ביחס ישר לטיב איכות המנגינה ביחס למדד זה: ככל שהציון גבוה יותר, הדירוג של המגינה גבוה יותר. המשקל היחסי מייצג את החשיבות של המדד לעומת שאר המדדים.

**דוגמה** – נתבונן במדד בסיסי הבודק עד כמה צלילי המנגינה "רציפים", במובן שהמרווחים בין זוגות צלילים סמוכים הם יחסית קטנים (למשל נמוכים מאיזשהו סף קבוע מראש). מתודת השערוך תבצע סריקה לינארית על רצף התווים של המנגינה, תחשב את המרווח בין כל זוג צלילים סמוכים ותמנה כמה מתוכן קפיצות תקינות וכמה חריגות. הציון שיוחזר מהשערוך יהיה היחס של מספר הקפיצות התקינות מתוך כלל הקפיצות (n-1 עבור רצף של n תווים). לבסוף ציון זה יוכפל במשקל היחסי שמוקצה למדד זה. באופן דומה מחושבים ציונים עבור כל יתר המדדים ולבסוף כלל הציונים מחוברים לסכום משוקלל שמהווה את הציון הכולל של המנגינה.

**מתודת ניהול השערוך**: המתודה EvaluateFitness היא המתודה האחראית לנהל את תהליך השערוך – היא עוברת על המדדים השונים, אומרת אותם ע"י הפעלת פונקציות השערוך המתאימות להם תוך התייחסות למשקליהם היחסיים, ולבסוף מסכמת את התוצאות לציון משוקלל. זוהי למעשה המתודה היחידה שמופעלת מתוך האלגוריתם הגנטי בהקשר השערוך. כל יתר המתודות נקראות מתוך המתודה EvaluateFitness.

**מחלקת ניהול המשקלים**: עבור המשקלים היחסיים, הוגדר סט משתנים קבועים עם ערכי ברירות מחדל. מאחר ששימוש במשקלים לשערוך הוא עניין תלוי מימוש שפנימי ללוגיקה שמנהלת את השערוך, משתני שיערוך אלו פחות שייכים למחלקה הראשית של האלגוריתם הגנטי. בכדי להימנע מ-"זיהום" המחלקה הראשית, וכן מתן מסגרת ל-DTO שיוכל לרכז ערכים מבוקשים עבור המשקלים ממשתמשי קצה דרך Web service למשל, הוגדרה מחלקת POCO ייעודית לריכוז המשקלים – המחלקה MelodyEvaluatorsWeights. מחלקה זו מרכזת את כלל המדדים עם משקלי ברירת המחדל שלהם, ערך ברירת מחדל לפקטור כללי שניתן להוסיף לכל המועמדים (לכל המנגינות).מחלקה זו מוגדרת גם היא בקובץ FitnessEvaluators.cs, המרכז את כלל פעילות השערוך. משמאל מוצגת דיאגרמת מחלקה שלה. היא פשוט מבנה בדומה ל-struct בשפת C/C++.

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

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

|  |  |
| --- | --- |
| Description | Method |
| 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. | EvaluateAccentedBeats |
| 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. | EvaluateContourDirection |
| 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. | EvaluateContourStability |
| 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 un pleasant drastic change in feel. | EvaluateDensityBalance |
| Evaluates the ratio of extreme pitch intervals. | EvaluateExtremeIntervals |
| Evaluates how "smooth" the movement is from tone to another. | EvaluateSmoothMovement |
| 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. | EvaluateSyncopation |
| 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. | EvaluatePitchRange |
| Evaluates fitness according to the variety of distinct pitches in use. | EvaluatePitchVariety |

#### מודול סלקציה (Selectors.cs)

**רקע**: סעיף זה מתאר את המודול האחראי על תהליך הסלקציה של המנגינות. מודול זה מרוכז בקובץ: Selectors.cs. המודול אחראי על הגדרת הלוגיקה לבחירת המנגינות שממשיכות לשלב הבא בכל מחזור (דור/איטרציה) של האלגוריתם. ישנן גישות שונות לבחירת המועמדים שממשיכים לשלב הבא, חלקן דטרמיניסטיות ומסתמכות אך ורק על הדירוג של המועמדים, ואילו אחרות אי-דטרמיניסטיות משלבות אלמנט של אקראיות 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 שונים. חלק מהשיטות משלבות כמה טכניקות שונות.

**מימוש הסלקציה במערכת**: בשלב זה המערכת מספקת שני מימושים שונים – האחד דטרמיניסטי, והשני אי-דטרמיניסטי –

* **המימוש הדטרמיניסטי** (PlusSelection) שהוגדר עובד בשיטה הנקראת "בחירת פלוס": חצי מהמקומות משוריינים למועמדים "צעירים" שנוצרו בסבב (דור/איטרציה) הנוכחי, ואילו יתר המקומות מוקצים למועמדים הוותיקים יותר מדורות קודמים. משני האוכלוסיות (צעירים וותיקים) מבוצעת בחירה של המועמדים הכשירים ביותר, דהיינו אלו בעלי הדירוג הגבוה ביותר. להלן שם המתודה:.
* **המימוש האי-דטרמיניסטי** (RouletteWheelSelection) שהוגדר, עושה שימוש במעין "רולטת מזל": לכל מועמד מוקצית רצועה על הרולטה, בהתאם ליחס שבין ציון טיב האיכות שלו לסכום הציונים של כלל המועמדים, ואז ע"י הגרלת מספר אקראי בטווח מוגדר ומיפוי המספר שיוצא לאחת מהרצועות ע"י מיפוין מראש על הטווח המוגדר, מבוצע דימוי של סיבוב מחוג מלאכותי, ונבחר המועמד שהמחוג "עצר" על הרצועה שלו. גם כאן יש עדיפות למועמדים כשירים יותר, שכן מועמדים כשירים יותר זוכים לרצועה גדולה יותר וכך מגדילים את ההסתברות שלהם לזכייה, אולם משולב גם אלמנט של אקראיות המקנה גם למועמדים אחרים הזדמנות. היתרון בהכנסת אקראיות ומתן הזדמנות גם למועמדים פחות כשירים הוא שזה מסייע בלמנוע מהחיפוש להיתקע על נקודות קיצון מקומיות, ומגדיל את מרחב החיפוש לרוחב, ולא רק לעומק. החיסרון הוא שזה כרוך בסיכון מסוים וכן שזמן ההתכנסות לפתרון איכותי מתארך. להלן שם המתודה:

המתודה **SelectNextGeneration** מספקת מעטפת למימושים השונים. זאת המתודה שהאלגוריתם הגנטי קורא לה בבקשה לבצע את הסלקציה/סינון, והיא בתורה אחראית לבצע delegation למימוש המתאים. נכון לעכשיו מבוצעת קריאה ישירה למתודת הבחירה הדרמיניסטית, אולם ניתן בקלות להגמיש את הבחירה המימוש ע"י הגדרת תלות של מתודת הבחירה הראשית (SelectNextGeneration) באיזשהו מנשק או אבסטרקציה שמייצגים אלגוריתם בחירה (אסטרטגיה), ולהזריק מימוש קונקרטי (Dependency Injection) תוך שימוש ב-Factory. אפשרות נוספת היא לשלב בין שיטת הבחירה השונות ולהחליף ביניהן לסירוגין , או לקבוע בכל מחזור (דור/איטרציה) את שיטת הבחירה מחדש ע"י "הטלת מטבע" בצורה של הגרלת מספר אקראי).

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

|  |  |
| --- | --- |
| Description | Method |
| 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). | SelectNextGeneration |
| 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. | PlusSelection |
| Filters out the current population of candidates with an un-deterministic algorithm that simulates a roulette wheel spin, and gives the candidates proportional chances to be selected according to their proportional fitness. | RouletteWheelSelection |

### הלחנה עם אלגוריתמים נוספים

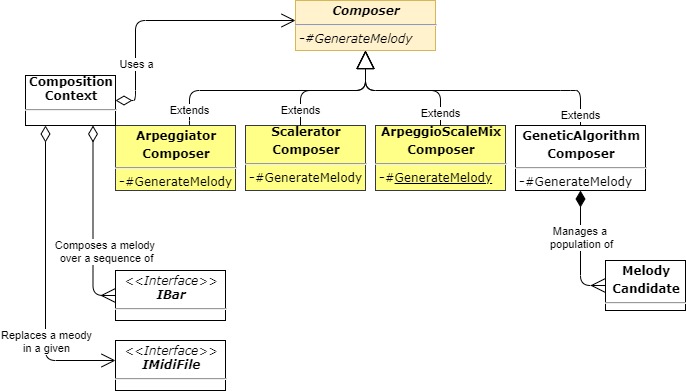
אלגוריתמי הלחנה במערכת מוגדרים במחלקות היורשות מהמחלקה האבסטרקטית [Composer](#_מחלקת_מלחין_אבסטרקטי). אלגוריתם ההלחנה המתוחכם ביותר במערכת ([האלגוריתם הגנטי](#_הלחנה_עם_אלגוריתם)) מוגדר במחלקה [GeneticAlgorithmComposer](#_GeneticAlgorithmComposer_Class). בנוסף למימוש אלגוריתם משוכלל זה, מוגדרים שלושה אלגוריתמים נוספים, בסיסיים ביותר, שפשוט מחוללים רצף תווים סטטי המתאים לרקע ההרמוני של היצירה. להלן המחלקות המממשות את האלגוריתמים הבסיסיים הללו ומרחב השמות שלהן –

|  |  |  |
| --- | --- | --- |
| Namespace | Class | # |
| CW.Soloist.CompositionService.Composers.Arpeggiator | ArpeggiatorComposer | 1 |
| CW.Soloist.CompositionService.Composers.Scalerator | ScaleratorComposer | 2 |
| CW.Soloist.CompositionService.Composers.ArpeggioScaleMix | ArpeggioScaleMixComposer | 3 |

שלושת האלגוריתמים שלעיל נבדלים ביניהם במקור סט התווים שבו הם משתמשים ליצירת רצף התווים. המחלקה הראשונה כוללת ברצף התווים שלה אך ורק תווים השייכים לאקורדים ביצירה, כלומר מול כל אקורד מתנגנים אך ורק צלילים המרכיבים אותו. המחלקה השנייה בוחרת תווים מתוך סולמות הממופים מול האקורדים, שלרוב כוללים את צלילי האקורד וצלילים נוספים, ואילו המחלקה השלישית מבצעת איזשהו שילוב.

שלושת המחלקות הללו פשוט דורסות את המתודה האבסטרקטית GenerateMelody ליצירת מנגינה, ובגוף המתודה הן משתמשות במתודות אתחול גנריות שמוגדרות במחלקת האב בכדי, ומפעילות אותן ישירות על רצף האקורדים (הפלייבק), כך שהאתחול מחולל את רצף התווים ו-"שותל" אותו לצד האקורדים ברצף התיבות שהתקבל בקלט לתיאור המהלך ההרמוני ומחזיר אותו.

הדיאגרמה שלהלן מדגישה את הקשר של מחלקות אלו בקונטקסט של תת-שכבה זו –



### מפעל ליצירת מלחינים

#### רקע

המערכת תומכת כרגע בארבעה אלגוריתמי הלחנה שונים. בכדי לא להיתלות במימוש ספציפי ולהקנות גמישות בשינויים עתידיים של אלגוריתם ההלחנה המבוקש בקלות, הוגדר סט קבועים (Enum) שבו מתוארים אלגוריתמי ההלחנה השונים שהמערכת תומכת בהם, ומחלקת מפעל שמקבלת קבוע המייצג אלגוריתם ומחזירה מופע של מחלקת מלחין המממשת את אסטרטגיית ההלחנה המבוקשת.

**המוטיבציה מאחורי סט הקבועים ומחלקת המפעל היא לשבור את התלות של קליינטים במימוש ספציפי של אלגוריתם הלחנה כזה או אחר, ולהשאיר תלות רק באבסטרקציה, דהיינו רק במחלקת האב האבסטרקטית** Composer**. קליינטים המעוניינים במופע של אלגוריתם הלחנה יפנו למחלקת המפעל בבקשה לקבלת מופע של מלחין, דהיינו של אלגוריתם הלחנה, והמחלקה תהיה אחראית לספק מופע שכזה, וכל זאת בלי שהקליינטים צריכים להכיר את המחלקות שמספקות את המימוש של האלגוריתם. זהו בעצם מימוש של עיקרון ה-**[Dependency Inversion Principle](https://en.wikipedia.org/wiki/Dependency_inversion_principle)**.**

#### **CompositionStrategy Enumeration**

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

**אוסף זה מאפשר לקליינטים ולמחלקת המפעל הייעודית לתקשר ביניהם בשפה משותפת על אלגוריתמי הלחנה ללא ציון שמות מחלקות קונקרטיות במפורש.**

**המערכת תומכת כעת כאמור בארבעה אסטרטגיות שונות. המוסכמה של שמות האסטרטגיות השונות היא** "<Algorithm Description>Strategy"**, כאשר** <Algorithm Description> **מוחלף בתיאור האלגוריתם. להלן אסטרטגיות ההלחנה הנתמכות בשלב זה (מוצגים גם בדיאגרמת מחלקה לעיל) –**

|  |  |
| --- | --- |
| Composing Strategy Name | **#** |
| ArpeggiatorStrategy | **1** |
| ArpeggioScaleMixStrategy | **2** |
| ScaleratorStrategy | **3** |
| GeneticAlgorithmStrategy | **4** |

#### **ComposerFactory Class**

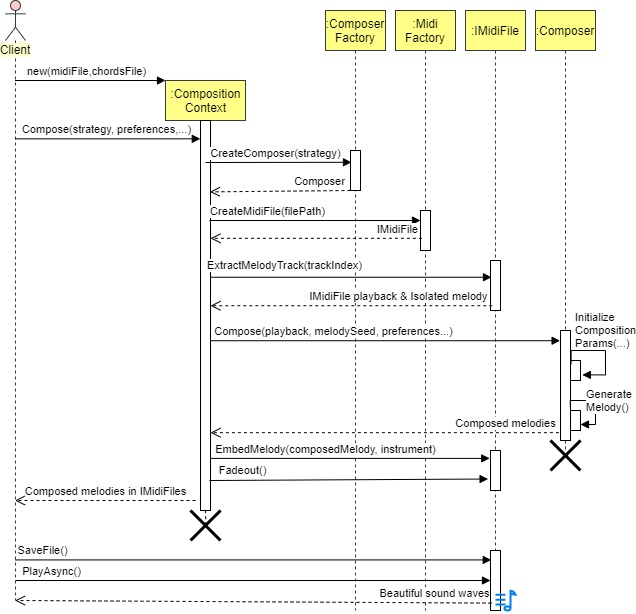
**המחלקה** ComposerFactory **היא מחלקה סטטית המספקת מתודה סטטית יחידה** CreateComposer **המקבלת כקלט אסטרטגיית הלחנה מבוקשת (**[CompositionStrategy](#_CompositionStrategy_Enumeration)**) ומחזירה מופע של** [Composer](#_מחלקת_מלחין_אבסטרקטי) **. היא מנהלת מיפוי בין אסטרטגיות הלחנה למחלקות המממשות אסטרטגיות אלו, עובדת גב אל גב ישירות מול מחלקות אלו ליצירת מופע שלהן, שאותו היא מחזירה לקליינטים מבלי שהם מודעים למחלקה הקונקרטית המוחזרת. מנגנון זה שובר תלות במימוש ומאפשר תחזוקה עתידית גמישה בהוספה ועדכון של אלגוריתמי הלחנה ומימושים שלהם.**

**להלן המיפויים המוגדרים בין אלגוריתמי הלחנה לבין המחלקות הממשות אותם, שמחלקת המפעל מחזירה מופע שלהן לקליינטים במסגרת המתודה** CreateCompositor **בהתאם לאסטרטגיית ההלחנה שהם מבקשים –**

|  |  |  |
| --- | --- | --- |
| Concrete Implementing Class | Composing Strategy | **#** |
| ArpeggiatorComposer | ArpeggiatorStrategy | **1** |
| ScaleratorComposer | ArpeggioScaleMixStrategy | **2** |
| ArpeggioScaleMixStrategy | ScaleratorStrategy | **3** |
| GeneticAlgorithmComposer | GeneticAlgorithmStrategy | **4** |

### אינטראקציה בין הישויות בתהליך הלחנה

להלן דיאגרמת רצף (Sequence diagram) הממחישה את האינטראקציה בין הישויות השונות בתת-שכבת ההלחנה ברצף אירועים טיפוסי שבו קליינטים מבצעים פנייה לשירות ההלחנה (קליינטים לדוגמה הם אפליקציית Web או Desktop) –



תיאור התרחיש בעמוד הבא.

**תיאור רצף האירועים בדיאגרמה שלעיל** –

1. קליינטים יוצרים מופע של CompositionContext על-סמך קובץ MIDI וקובץ אקורדים של השיר הנדרש ומבקשים מהקונטקסט להלחין מנגינה חדשה עם המתודה Compose, תוך העברת פרמטרים נוספים כגון אסטרטגיית אלגוריתם ההלחנה המבוקשת, העדפות אישיות, אילוצים ואינדקס רצועת המנגינה בקובץ ה-MIDI שהועבר.
2. קונטקסט ההלחנה לוקח את המשוכות מכאן ומטפל מול כל יתר הגורמים הרלוונטיים:
   1. ראשית הוא פונה למפעלים לקבלת מופעים של IMidiFile ו-Composer (על סמך האסטרטגיה המבוקשת).
   2. לאחר מכן הוא מבודד את רצועת המנגינה מתוך קובץ ה-MIDI ע"י פנייה במתודה ExtractMelody למופע ה-IMidiFile, כך שכעת יש בידיו קובץ פלייבק (ה-MIDI ללא המנגינה) ואת המנגינה בנפרד, שיכולה לשמש כגרעין השראה לבסס עליו את האתחול.
   3. עם נתונים מעובדים אלו והפרמטרים הנוספים מהקליינט הקונטקסט פונה למלחין בבקשה להלחנה עם המתודה Compose וממתין לתוצאות.
3. המלחין ראשית מבצע אתחול ולאחר מכן קורא למתודה הפרטית שלו GenerateMelody שמבצעת את תהליך ההלחנה בפועל. זוהי כאמור מתודה אבסטרקטית במחלקה Composer שמחלקות המלחינים הקונקרטיים אחראים לממש. בסיום העיבוד, המתודה מחזירה לקונטקסט ההלחנה אוסף של מנגינות.
4. למען פשטות – התרחיש בדיאגרמה מתאר מצב שבו הקליינט מעוניין במנגינה אחד בלבד מבין אלו שחזרו (למשל זו הראשונה). הקונטקסט משבץ את המנגינה הנבחרת בקובץ MIDI של הפלייבק (הקובץ ללא מנגינה) עם המתודה EmbedMelody. במידה ויש עניין ביותר ממנגינה אחת הקונטקסט יוצר פלייבקים נוספים עם המתודה CreatePlayback של IMidiFile וחוזר על התהליך עבור כל אחת מהמנגינות.
5. לבסוף הקונטקסט מחליש את הווליום בתיבות האחרונות בקובצי ה-MIDI ומחזיר אותם יחד עם המגינות החדשות שהולחנו בחזרה לקליינט בדמות של מערך של IMidiFile.
6. הקליינט יכול לבקש ממופעי ה-IMidiFile להתנגן עם המתודה Play (או PlayAsync) ו/או לשמור עותק פיזי על הדיסק הקשיח של קובץ ה-MIDI המיוצג ע"י מופע ה-IMidiFile.
7. עם קובץ MIDI שמור הקליינט יכול להיעזר בתוכנות צד ג' בשביל להמשיך לעבד את קובץ ה-MIDI – להמיר אותו למסמך תווים להדפסה, לנגן אותו, לשנות לו את גובה הצליל, וכד'.

### הרחבות (ComposerExtensions)

המחלקה ComposerExtensions כשמה כן היא, מגדירה אוסף של Extension Methods עבור שירותי הלחנה, המרחיבות פונקציונאליות עבור טיפוסים של רצפים ואוספים של תווים, דוגמת מיון וערבול. מתודות אלו שימושיות ליצירת פרמוטציות שונות של תווים במסגרת תהליך ההלחנה. להלן המתודות המוגדרות במחלקה זו –

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

### אוספי קבועים (Enums)

מרחב השמות CW.Soloist.CompositionService.Enumsמגדיר אוסף של Enumerations המגדירים אוספי עזר של קבועים שנמצאים בשימוש תדיר בתת-שכבת הלחנת המנגינות, כגון קטגוריות פרמטרים למתודות המשפיעות על אופן הביצוע. כמו כן מוגדרת במרחב שמות זה מחלקת הרחבות כללית עבור Enums. בסעיף זה מפורטים כל אוספי קבועים אלו ומחלקת ההרחבות.

#### **EnumExtensions**

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

|  |  |  |
| --- | --- | --- |
| Description | The Subject Entity | Method |
| Returns a friendly string representation for the given enum value, which is defined for it in a custom DescriptionAttribute annotation. | Enum | GetDescription |
| 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. | Enum | GetDisplayName |

#### **ChordNoteMappingSource**

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

|  |  |
| --- | --- |
| Description | Name |
| 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). | Chord |
| 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. | Scale |

#### **DurationSplitRatio**

אוסף זה מגדיר קבועים המייצגים יחס לפיצול משך שהייה –

|  |  |
| --- | --- |
| Description | Name |
| 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. | Equal |
| Represents an unequal ratio split of 1:3. The first duration after the split would be 1/4 long in relation to the original duration, and the second duration would be 3/4 long in relation to the original duration.  For example, if the original duration was 1/2, then after the split the first duration would be 1/8, and the second duration would be 3/8. | Anticipation |
| Represents an unequal ratio split of 3:1. The first duration after the split would be 3/4 long in relation to the original duration, and the second duration would be 1/4 long in relation to the original duration.  For example, if the original duration was 1/2, then after the split the first duration would be 3/8, and the second duration would be 1/8. | Delay |

#### **NoteSequenceMode**

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

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

#### **OverallNoteDurationFeel**

אוסף זה מגדיר קבועים המייצגים משך שהייה שיינתן למרבית התווים, מה שמשפיע על רמת הדחיסות הכוללת של צלילים במנגינה וקובע את תחושת הקצב הכללית –

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

#### **Permutation**

אוסף זה מגדיר קבועים המייצגים פרמוטציות אפשריות שונות לרצפי תווים –

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

#### **PitchRangeSource**

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

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

#### **SortOrder**

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

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

# שכבת הגישה לנתונים (DAL)

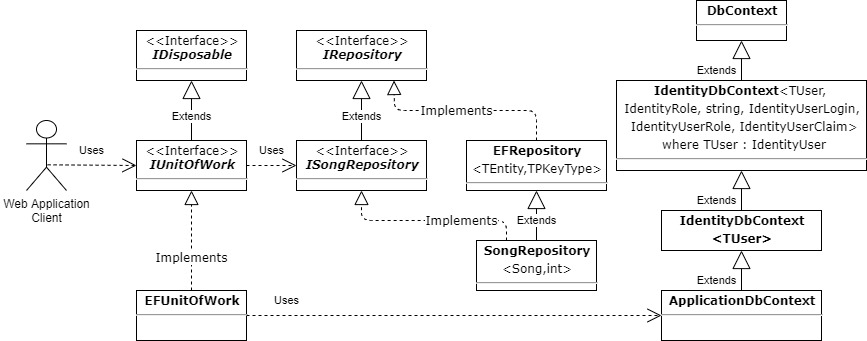
שכבה זו של ה-DAL (Data Access Layer) מטפלת בכל הלוגיקה הנדרשת לגישה לנתונים – חיבור המערכת למסד נתונים פיזי, ניהול הגישה למסד נתונים זה מתוך המערכת, אחזור ועדכון של הנתונים, המרת הנתונים מאופן ייצוגם במדיום האחסון הפיזי לאופן ייצוגם התואם במערכת בעולם האובייקטים של התכנות מונחה עצמים (ולהפך), ואספקת אבסטרקציות ושירותי גישה עוטפים לקליינטים כך שיוכלו לגשת לנתונים על סמך חוזה התקשרות מוסכם ללא תלות באופן המימוש הפיזי של הגישה וללא תלות במדיום הפיזי שבו הנתונים נשמרים בפועל מאחורי הקלעים.

השכבה מוגדרת בשלמותה כספרייה בפרויקט ייעודי המהווה יחידת אסמבלי עצמאית: DataAccess. בהתאם (וע"פ [מוסכמות השמות במערכת למרחבי שמות](#NamespaceConventions)), כלל מרחבי השמות (namespaces) השייכים לשכבה זו מושרשים תחת מרחב השמות: CW.Soloist.DataAccess, המחולק לרכיבים לוגיים שונים תחת מרחבי שמות נפרדים, כאשר כל רכיב אחראי על השלמת פונקציונליות/שירות מסוים. להלן סקירה קצרה של הרכיבים השונים המרכיבים את שכבת הגישה לנתונים עם מרחב-השמות הפנימיים שלהם (בעמודים הבאים רכיבים אלו מתוארים בפירוט בחלוקה לפי מרחבי השמות הבאים) –

|  |  |  |
| --- | --- | --- |
| # | תפקיד תת-הרכיב בשכבת הגישה לנתונים | שם מרחב שמות (Namespace) של הרכיב |
| 1 | אספקת שירות ניהול גישה לנתונים לקליינטים באמצעות הגדרת אבסטרקציה (ומימוש) של מנהל ישויות המתפקד כ-Unit of Work שאחראי על מעקבי שינויים ואינטראקציה בין הרכיבים השונים בשכבה זו. | CW.Soloist.DataAccess.UnitOfWork |
| 2 | ריכוז כל המודלים – הישויות העסקיות במערכת הרלוונטיות לאחסון (Persistence). | CW.Soloist.DataAccess.DomainModels |
| 3 | הגדרת אבסטרקציות לגישור בין המודלים (ישויות עסקיות הרלוונטיות ל-Persistence) לבין מדיום אחסון הנתונים הפיזי, וריכוז מימושים של אבסטרקציות אלו. | CW.Soloist.DataAccess.Repositories |
| 4 | ריכוז קונפיגורציות וישויות המספקות שירותי ORM – שירותי מיפוי בין אובייקטים לרשומות בבסיסי נתונים רלציוניים באמצעות שירותי הספרייה של EntityFramework. | CW.Soloist.DataAccess.EntityFramework |
| 5 | ריכוז מיגרציות (עדכונים) על הישויות ובסיס הנתונים שנמצאים בפיקוח של ה-EntityFramework במטרה להביא אותם ליישור קו בתאימות גרסאות עם יכולת Rollback. | CW.Soloist.DataAccess.Migrations |

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

להלן דיאגרמת מחלקה המתארת את הקשרים בין הרכיבים המרכזיים בשכבת הגישה לנתונים –



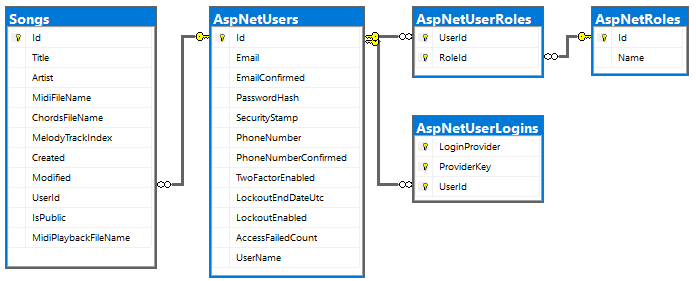
כפי שצוין לעיל, היישום שמשתמש בשכבת הגישה לנתונים מנהל אינטראקציה אך ורק עם מנשק ה-IUnitOfWork, כך שאם בעתיד יידרש להחליף את המימוש הפנימי של שכבה זו, כל עוד שהמימוש החדש ימשיך לעמוד בחוזה הכללי שמוגדר ב-IUnitOfWork (חוזה שאינו תלוי ב-Entity Framework או פלטפורמה ספציפית אחרת), ההחלפה תהיה שקופה לחלוטין לקליינטים, למעט אולי עדכון חד-פעמי של רישום התלויות חד-פעמית בקונפיגורציה של ה-IoC להיפוך והזרקת התלויות (Inversion Of Control) המוגדרת באמצעות ספריית AutoFac.

## מסד הנתונים (Database)

המערכת מאחסנת את הנתונים שמשמשים את האפליקציה בבסיס נתונים רלציוני. בסעיף זה מפורט מבנה בסיס נתונים זה.

### דיאגרמת טבלאות

להלן דיאגרמה המתארת את הקשרים בין הטבלאות השונות בבסיס הנתונים –



### פירוט הטבלאות

להלן תקציר הטבלאות המרכזיות בבסיס הנתונים –

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

בתת-סעיפים הבאים מפורטות הסכמות של כל אחת מטבלאות אלו.

#### **AspNetUserLogins Table**

**טבלת זו מכילה פרטי חשבונות חיצוניים של אפליקציות צד ג' של משתמשים רשומים, דוגמת** Google**,** Microsoft**, או** Linkedin**. חשבונות חיצוניים אלו יוכלו לאמת את המשתמש אל מול היישומים החיצוניים באמצעות פרוטוקול הזדהות כגון** OAuth**. להלן שדות הטבלה –**

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

#### **AspNetRoles Table**

**טבלת זו מכילה את הרולים (תפקידים) במערכת המשמשים לניהול הרשאות משתמשים רשומים. מוגדרים שני רולים – רול** Admin **של מנהלן מערכת, ורול** ApplicationUser **עבור כל יתר המשתמשים הרשומים –**

|  |  |  |
| --- | --- | --- |
| Description | Field Type | Field Name |
| Internal id of the role in the database. | nvarchar(128) | Id |
| Role name. | nvarchar(256) | Name |

#### **AspNetUsers Table**

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

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

#### **AspNetUserRoles Table**

**טבלת זו היא טבלת קשר הממפה בין משתמשים לבין הרולים המוקצים להם -**

|  |  |
| --- | --- |
| Field Type | Field Name |
| nvarchar(128) | UserId |
| nvarchar(128) | RoleId |

#### **Songs Table**

טבלת השירים מכילה את נתוני כל השירים שמשתמשי האפליקציה (לרבות מנהלן המערכת) העלו למערכת. השירים עצמם (דהיינו קבצי ה-Midi וקובצי האקורדים של השירים) אינם מאוחסנים בבסיס הנתונים עצמו בטבלה ישירות, אלא על שרת הקבצים (File server). בטבלת השירים נשמרים רק הנתיבים לקובצי השירים על שרת הקבצים.

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

## מודלים (Domain Models)

מודלים (Domain Models) הם מחלקות עבור ישויות עסקיות במערכת שרלוונטיות ל-Persistence, דהיינו מחלקות שרלוונטיות לאחסון בבסיס נתונים). מחלקות אלו הן מחלקות POCO (Plain Old CLR Objects), דהיינו מחלקות פשוטות שלא נדרש מהן פונקציונאליות מיוחדת שמגיעה בירושה או מימוש מנשק כזה או אחר, אלא רק הגדרת סט מאפיינים של הישות ושגרות בסיסיות לעדכון ואחזור מאפיינים אלו. המודלים מספקים ייצוג בעולם האובייקטים לטבלאות השטוחות הקיימות בבסיס הנתונים – בד"כ מופע של מודל מייצג איזושהי רשומה בטבלה בבסיס הנתונים שממופה מול אותו מודל.

בנוסף למאפיינים שמוגדרים במחלקות מודלים אלו כנגד השדות בטבלאות ה-Database, המודלים מגדירים גם מאפיינים לניווט (Navigation Properties) בין ישויות מקושרות כנגד הקשרים שבין הטבלאות. למשל – עבור קשר של רבים–לאחד מטבלת השירים אל טבלת המשתמשים, המיוצג בבסיס הנתונים ע"י אילוץ מפתח זר של מזהה המשתמש ברשומות של השירים, במחלקת המודל המייצגת שירים יהיו שני מאפיינים עבור הקשר למשתמש –מאפיין אחד עבור מזהה המשתמש שהעלה את השיר כנגד העמודה התואמת בטבלה ב-Database, ומאפיין שני עבור מופע תואם של אותו משתמש המחלקה שמייצגת משתמשים. באופן דומה מהצד השני של הקשר, יהיה מוגדר מאפיין ניווט של רשימה שמכיל את רשימת כל המופעים של שירים שהועלו ע"י אותו משתמש.

כלל המודלים במערכת מוגדרים במרחב השמות הייעודי הבא –

CW.Soloist.DataAccess.DomainModels

סעיף זה מפרט את מחלקות המודלים השונות.

### מודל משתמש (ApplicationUser Model)

המחלקה ApplicationUser מייצגת משתמש במערכת. מחלקה זו יורשת מהמחלקה IdentityUser הקיימת ב-.NET, שמכילה פונקציונאליות מובנית לתמיכה ב-[AAA](https://he.wikipedia.org/wiki/AAA_(%D7%AA%D7%A7%D7%A9%D7%95%D7%A8%D7%AA_%D7%9E%D7%97%D7%A9%D7%91%D7%99%D7%9D)) – Authentication, Authorization & Accounting. מחלקה זו אחראית גם על הייצוג באובייקטים כנגד הטבלאות המתאימות להן בבסיס הנתונים – AspNetUsers, AspNetUserLogins וכד'.

מעבר להגדרות המובנות של .NET, נוספה למחלקה זו הגדרת מאפיין ניווט עבור הקשר שבין משתמשים לשירים. בכל מופע של משתמש, מוגדרת רשימת כל השירים שהוא העלה למערכת –

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

#### שמות רולים (**RoleName**)

המחלקה 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, מאחר שהוא נוגע ישירות למאפיינים של מודל המשתמש.

#### **AuthorizationActivity Enumeration**

אוסף הקבועים (Enum) AuthorizationActivity מגדיר סוגי פעילות הרלוונטיים למידור ובדיקת הרשאות. האוסף מוגדר בקובץ ApplicationUser.cs יחד עם המחלקה ApplicationUser, מאחר שהוא נוגע ישירות למאפיינים של מודל המשתמש. הקבועים המוגדרים הם כדלקמן –

public enum AuthorizationActivity {

Create = 1,   Update = 2,

Display = 3, Cancel = 4,   Delete = 5 }

### מודל שיר (Song Domain Model)

המחלקה Song מייצגת ישות/רשומה של שיר, כנגד הטבלה Song בבסיס הנתונים.

להלן רשימת המאפיינים המוגדרים במחלקה זו –

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

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

public void SetPlaybackName(string referenceName = null) { … }

#### **SongFileType Enumeration**

אוסף הקבועים (Enum) SongFileType מגדיר סוגי קבצים שמנוהלים באפליקציה – קובץ אקורדים – קובץ טקסט, קובץ MIDI של השיר המקורי כפי שהועלה למערכת וקובץ פלייבק – קובץ ה- MIDI לאחר שהוסרה ממנו רצועת המנגינה. בהתאמה, ערכים 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.

## [מאגרים (Repositories)](https://github.com/cwelt/Soloist/tree/master/DataAccess/Repositories)

כלל רכיבי הפיתוח בסעיף זה מוגדרים במרחב השמות הבא –

CW.Soloist.DataAccess.Repositories

קבצי קוד המקור המכילים רכיבים אלו נגישים בקישור הבא –

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

מאגר ([Repository](https://martinfowler.com/eaaCatalog/repository.html)) הוא דפוס עיצוב ארכיטקטוני שמגשר בין ישויות עסקיות (מודלים) לבין המיפוי שלהם מול בסיס הנתונים הפיסי, מעין קופסא שחורה שעוטפת את הלוגיקה של המיפוי, המספקת לקליינטים יכולת לבצע אינטראקציה מול מדיום-האחסון באופן מופשט, High-level,

בטרמינולוגיה של הישויות העסקיות בעולם האובייקטים, מבלי הצורך להתעסק או לדעת כיצד הישויות מאוחסנים בפועל (טבלאות, עמודות וכו').

ה-Repository כשמו כן הוא, מאגר. ככזה, הוא מנהל אוסף של ישויות במאגר, לרוב מאותו טיפוס, ע"י הגדרת פונקציונאליות לעבודה מול המאגר – הוספת וגריעה של ישויות אל/מ המאגר, ומילוי המאגר מתוך מסד הנתונים.

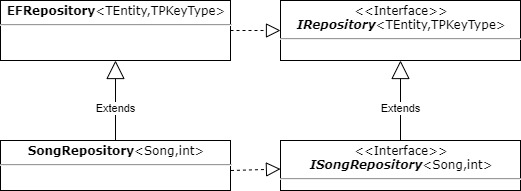
לאחר שמירת שינויים שבוצעו על ישויות במאגר בעולם האובייקטים, השינויים התואמים עוברים לביצוע בפועל במסד הנתונים – ישויות שנגרעו מהמאגר גורמות למחיקת רשומות מהטבלאות התואמות בבסיס הנתונים, ישויות חדשות שנוספו למאגר נוספות כרשומות חדשות טבלאות וכן הלאה.

לשם כך ה-Repositories מגדירים פעולות מילון – Create, Read, Update, Delete CRUD –, ושאילתות כגון – אחזור ישות לפי מזהה חד-ערכי (מפתח) או אחזור אוסף כל הישויות העונות לסט קריטריונים. כלל הפעולות האלה מוגדרות ב-Repositories כמתודות, ובגוף המימוש של מתודות אלו מבוצעת כל הלוגיקה של ה-Low-level של תרגום הקריטריונים ומאפיינים מעולם אובייקטים לטרמינולוגיה וייצוג בעולם בסיסי הנתונים הרלציוניים – ה-Repository דואג לתרגם את המתודות לשאילתות מתאימות ב-SQL, ולאחר ביצוע שלהם בבסיס הנתונים, מעדכן את המאגר עם מופעי הישויות בזמן ריצה בהתאם לסטטוס העדכני שלהם בבסיס הנתונים.

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

הטמעת דפוס ה-Repositories במערכת מבוצעת ע"י הגדרת Repository ייעודי כנגד כל ישות עסקית (מודל). אם למשל יש טבלת שירים במסד הנתונים בשם Songs וישות עסקית של מודל המייצג שיר במחלקה הנקראת Song, נגדיר Repository ייעודי, למשל SongRepository שיהיה אחראי על אינטראקציה מול בסיס הנתונים בהקשר של שירים. אם נרצה למשל לשמור במסד הנתונים גם נתוני פרופיל משתמש למשל, נקים טבלה מתאימה ב-Database, נגדיר מחלקת מודל כנגדה, ואז נגדיר Repository ייעודי (למשל UserProfileRepository) שיהיה אחראי על אינטראקציה מול ישויות נתוני פרופיל משתמש.

להלן דיאגרמת מחלקה המתארת את ישויות ה- Repositoryהשונות המרכיבות מודול זה של המאגרים, ואת הקשרים שביניהם –



### [מנשק מאגר (IRepository Interface)](https://github.com/cwelt/Soloist/blob/master/DataAccess/Repositories/IRepository.cs)

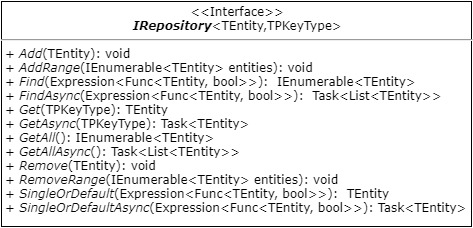
המנשק IRepository מגדיר מנשק-אב גנרי עבור כלל מנשקי ה-Repository –

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

ב-Repositories הקונקרטיים, הפרמטר הגנרי TEntity מוחלף בטיפוס הישות העסקית של אותו Repository – טיפוס מחלקת המודל הרלוונטית, והפרמטר הגנרי TPKeyType מוחלף בטיפוס המאפיין/שדה באותה מחלקת מודל 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 במערכת. אם בעתיד יידרש להגדיר ייצוג תואם גם עבור ישות שהזיהוי החד-ערכי שלה מורכב מ-2 שדות או יותר (ולא משדה יחיד), עדיין ניתן יהיה לעשות שימוש בתצורה הנוכחית ע"י שרשור השדות השונים המרכיבים את המפתח במחלקת המודל של ישות עסקית זו לשדה ייעודי חדש, מטיפוס String, שייצג את המפתח בשדה בודד, ואז ב-Repository לפרק שדה זה בחזרה לשדות המרכיבים, ואיתם לגשת לטבלאות בבסיס הנתונים. מימוש גנרי יותר יספק תמיכה בקבלת אוסף עמודות המייצגות את המפתח, דוגמת מערך ה-DataColumn שאובייקט ה-DataTable מקבל כפרמטר למפתח בגישת ה-Disconnected Layer של ADO.NET.

להלן דיאגרמת מחלקה עבור המנשק IRepository המתארת את מתודות המנשק –



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

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

מתודות הוספה ומחיקה של ישויות מהמאגר –

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

עבור כל ישות קונקרטית שרלוונטית ל-Persistence, יוגדר Repository ייעודי היורש ממנשק-האב הגנרי עם טיפוסים קונקרטיים. המנשקים הייעודים יוכלו להעשיר את המנשק הבסיסי בפונקציונאליות פרטנית יותר, למשל, מאגר שירים עשוי להגדיר מתודה שמאפשרת לאחזר שירים ע"פ שם המחבר/להקה, או אחזור אוסף כל השירים המסווגים תחת סגנון מוסיקלי מסוים.

קוד המקור של המנשק מוגדר בקובץ IRepository.cs שניתן למצוא בקישור הבא –

<https://github.com/cwelt/Soloist/blob/master/DataAccess/Repositories/IRepository.cs>

### [מנשק מאגר שירים (ISongRepository Interface)](https://github.com/cwelt/Soloist/blob/master/DataAccess/Repositories/ISongRepostiory.cs)

המנשק ISongRepository יורש ממנשק-האב הגנרי [IRepository](#_מנשק_מאגר_(IRepository) ומגדיר מנשק מאגר עבור ישויות של שירים, המיוצגים ע"י מחלקת המודל [Song](#_Song_Domain_Model) –

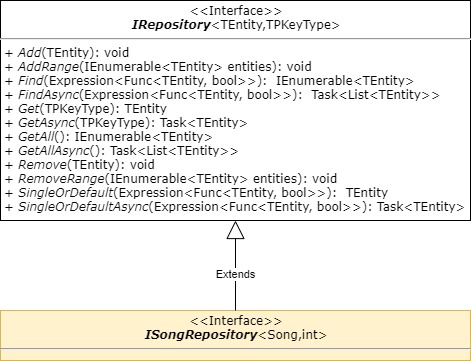
public interface ISongRepostiory : IRepository<Song, int> { }

לפי שעה מנשק זה משמש כמנשק תיוג, המגדיר את הטיפוסים הקונקרטיים של הישות העסקית שמאגר זה אחראי עליה – Song, ושל המפתח – המזהה החד-ערכי של ישות זו – int. אם יידרשו מתודות למניפולציה על מאגר השירים, כגון אחזור על השירים לפי סגנון מוסיקלי מסוים, הן יוגדרו במנשק זה.

קוד המקור של המנשק מוגדר בקובץ IRepository.cs שניתן למצוא בקישור הבא –

<https://github.com/cwelt/Soloist/blob/master/DataAccess/Repositories/ISongRepostiory.cs>

להלן דיאגרמת מחלקה עבור המנשק ISongRepository –



### [מחלקת-אב גנרית למימוש מנשק המאגר הגנרי (EFRepository Class)](https://github.com/cwelt/Soloist/blob/master/DataAccess/Repositories/EFRepository.cs)

המחלקה EFRepository מגדירה מחלקת-אב גנרית המממשת את מנשק-האב הגנרי [IRepository](#_מנשק_מאגר_(IRepository):

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

ה-EF בשם המחלקה הם ראשי-תיבות של [Entity Framework](https://docs.microsoft.com/en-us/ef/ef6/), חבילת ה-[ORM](https://he.wikipedia.org/wiki/%D7%9E%D7%99%D7%A4%D7%95%D7%99_%D7%90%D7%95%D7%91%D7%99%D7%99%D7%A7%D7%98%D7%99-%D7%A8%D7%9C%D7%A6%D7%99%D7%95%D7%A0%D7%99) מבית Microsoft. שם המחלקה מרמז על האופן שבו היא מממשת את המנשק IRepository: באמצעות הפונקציונאליות שכבר מסופקת "Out of the Box" ע"י חבילת ה-EF: החבילה מגדירה טיפוס DbContext המתפקד כמנהל ישויות תוך מימוש הדפוס הארכיטקטוני [Unit of Work](https://martinfowler.com/eaaCatalog/unitOfWork.html), ומחזיק אוסף ישויות גנריות מטיפוס DbSet, אשר כל אחת מהן מתפקדת למעשה כ-Repository עבור הישות מהטיפוס הקונקרטי של אותו DbSet. לפיכך, במקום לכתוב מאפס את כל הלוגיקה למיפוי של ישות עסקית מול טבלת DB, המרת מתודות לשאילתות SQL וכו', ניתן בגוף מימוש מתודות המנשק פשוט לבצע האצלת אחריות (delegation) אל ישויות ה-EF.

היתרון בשימוש במנשק ובמחלקה מגשרת זו במקום בשימוש ישיר ב-EF בקוד בצד הקליינטים (יתר שכבות המערכת) הוא האבסטרקציה: הקליינטים אינם תלויים ב-EF אלא רק במנשק המופשט של ה-IRepository. התלות באבסטרקציה מאפשרת להחליף את המימוש הקיים של מחלקה זו במחלקה אחרת, שמממשת את המנשק ע"י חבילת ORM אחרת (למשל [NHibernate](https://nhibernate.info/)), או בכלל בשימוש ישיר של ספריית ה-ADO.NET. כל עוד המימוש הוא של המנשק IRepository, תוך שימוש ב-Factory מתאים (או מנגנון Dependency Injection), לקליינטים שעובדים מול המנשק בלבד המחלקה הקונקרטית שמממשת את המנשק שקופה לחלוטין.

מחלקה זו היא כאמור גנרית. עבור הישויות השונות המעוניינות לעשות שימוש במחלקה זו למימוש מאגר יש להגדיר מחלקות קונקרטיות היורשות ממחלקה זו, ולהחליף את הטיפוס הגנריים בטיפוס הקונקרטיים של המודל הרלוונטי וטיפוס המאפיין שמשמש כמזהה החד-ערכי של המודל.

להלן המאפיינים/שדות המוגדרים במחלקה זו: מאפיין מטיפוס DbContext שאחראי כאמור על ניהול הישויות ואינטראקציה מול DB, ושדה מטיפוס גנרי DbSet<TEntity> שמחלקות מאגר קונקרטיות צריכות להחליף עם הטיפוס של ישות המודל הקונקרטית –

protected DbContext Context { get; }

private readonly DbSet<TEntity> \_entities;

המחלקה מגדירה בנאי יחיד שדרכו ניתן "להזריק" תלויות במאפיינים שלעיל כארגומנטים –

public EFRepository(DbContext context)

{

Context = context;

\_entities = context.Set<TEntity>();

}

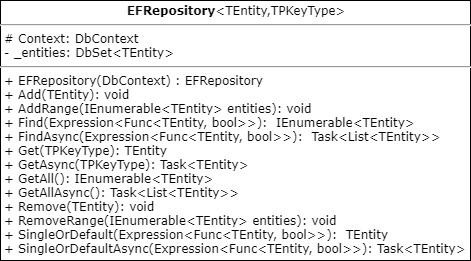
להלן דוגמה לאופן ביצוע האצלת הסמכות (delegation) למימוש הקיים ב-EF: המתודה Get המוגדרת במנשק IRepository עוברת אדפטציה למתודה Find של ישות ה- DbSet –

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

קוד המקור המלא של המחלקה מוגדר בקובץ EFRepository.cs שניתן למצוא בקישור הבא –

<https://github.com/cwelt/Soloist/blob/master/DataAccess/Repositories/EFRepository.cs>

להלן דיאגרמת מחלקה עבור המחלקה EFRepository המתארת את רכיבי המחלקה –



### [מחלקת מאגר שירים (SongRepository Class)](https://github.com/cwelt/Soloist/blob/master/DataAccess/Repositories/SongRepository.cs)

המחלקה SongRepository יורשת ממחלקת-האב הגנרית [EFRepository](#_מחלקת-אב_גנרית_למימוש) ומגדירה מחלקת מאגר עבור ישויות של שירים, המיוצגים ע"י מחלקת המודל [Song](#_Song_Domain_Model) –

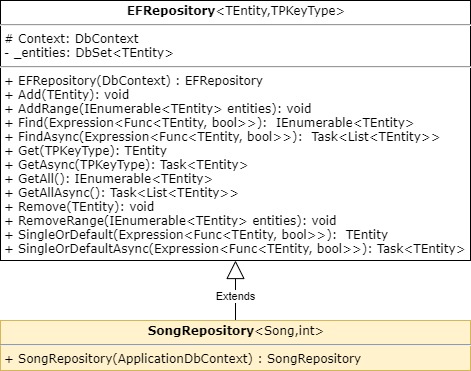
public class SongRepository : EFRepository<Song, int>, ISongRepostiory

לפי שעה מנשק מחלקה זו אינה מרחיבה את הפונקציונאליות הקיימת במחלקת האב, אלא רק מגדירה את הטיפוסים הקונקרטיים במקום אלו הגנריים.

קוד המקור של המחלקה מוגדר בקובץ SongRepository.cs שניתן למצוא בקישור הבא –

<https://github.com/cwelt/Soloist/blob/master/DataAccess/Repositories/SongRepository.cs>

להלן דיאגרמת מחלקה עבור המחלקה SongRepository המתארת את רכיבי המחלקה וקשר הירושה שלה למחלקת-האב הגנרית EFRepository–



## [ניהול יחידות עבודה (Unit Of Work)](https://github.com/cwelt/Soloist/tree/master/DataAccess/UnitOfWork)

כלל רכיבי הפיתוח בסעיף זה מוגדרים במרחב השמות הבא –

CW.Soloist.DataAccess.UnitOfWork

קבצי קוד המקור המכילים רכיבים אלו נגישים בקישור הבא –

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

יחידת עבודה ([Unit of Work](https://martinfowler.com/eaaCatalog/unitOfWork.html)) היא דפוס עיצוב ארכיטקטוני לניהול עדכוני זמן ריצה על אוסף ישויות עסקיות הרלוונטיות ל-Persistence וסנכרון שינויים אלו אל מול מדיום האחסון הפיסי, וכל זאת באמצעות מנשק High-level, בטרמינולוגיה של הישויות העסקיות בעולם האובייקטים, מבלי הצורך להתעסק או לדעת כיצד הישויות מאוחסנים בפועל (SQL/NoSQL/XML וכו').

ה- Repositoryאחראי על עדכון ניהול ישויות במאגר בזיכרון זמן הריצה, וה- Unit of Workהוא זה שאחראי לסנכרן את תמונת המצב בין מאגר/ים שבזיכרון זמן ריצה אל מול תמונת המצב במדיום האחסון הפיסי. כמו כן, אם ה-Repository מנהל בד"כ אוסף של ישויות במאגר מאותו טיפוס, ה-Unit of Work מנהל אוסף של אוספי ישויות, דהיינו אוסף של Repositories.

לשם כך ה-Unit Of Work מגדיר פעולות לרישום ומעקב סטטוס הישויות השונות, ופעולות לשמירת אוסף השינויים שהצטברו בזמן ריצה אל מול מדיום האחסון הפיסי. למשל פעולת רישום ישויות "מלוכלכות" – דהיינו רישום מופעים של אובייקטים בזמן ריצה חדשים ו/או מופעים שעודכנו מאז שנקראו ממדיום האחסון הפיסי, מודיעה ל- Unit Of Workאילו מהישויות יש לדאוג לסנכרן בחזרה אל מול מדיום האחסון הפיסי בזמן שמירה.

מנקודת המבט של הקליינטים שעובדים עם ה-Unit Of Work, הם רק צריכים לקרוא למתודות high-level כגון Commit, מבלי הצורך לדעת האם Commit זה מבצע מאחורי הקלעים פקודת SQL (Insert/Update/Delete), או בכלל עדכון איזה קובץ XML.

מימוש דפוס ה-Unit Of Work במערכת בוצע תוך ניצול המימוש הקיים בחבילת ה-ORM של .NET – EntityFramework[[2]](#footnote-2): ישויות ה-DbSet של ה- EntityFrameworkמתפקדות בתור Repositories, וישות ה-DbContext מתפקדת בתור UnitOfWork. לפיכך, ניתן לקחת קריאות ל-UnitOfWork ולהמיר אותן (להתאים אותן) לקריאות שה- DbContextכבר מכיר ומבצע היטב, ובעצם להאציל אליו את האחריות. הערה: במערכת לא נעשה שימוש ישיר ב- DbContextאלא במחלקה ApplicationDbContext היורשת ממנה, ומרחיבה את הפונקציונאליות שלה עם שירותים שמותאמים ליישום Web-י וניהול רשומות משתמש.

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



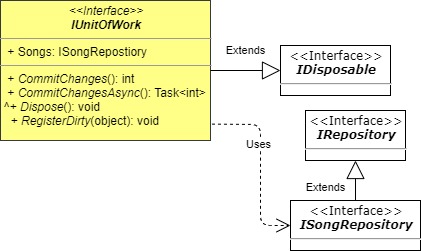
### [מנשק יחידת עבודה (IUnitOfWork Interface)](https://github.com/cwelt/Soloist/blob/master/DataAccess/UnitOfWork/IUnitOfWork.cs)

המנשק IUnitOfWork מגדיר מנשק עבור הפשטת פונקציונליות דפוס יחידת עבודה –

public interface IUnitOfWork : IDisposable {…}

מנשק זה הוא למעשה נקודת הגישה של הקליינטים ל-Database. הקליינטים פונים למנשק ה-UnitOfWork לקבלת גישה למאגרים המנוהלים אצלו כ-Properties, מעדכנים את המאגרים באמצעות המנשק שהם מספקים לאחזור ועדכון ישויות בזיכרון זמן ריצה, ולבסוף מבקשים מה-UnitOfWork לסנכרן את תמונת המצב של המאגרים שהוא מכיל אל מול תמונת המצב ב-Database.

להלן דיאגרמת מחלקה עבור המנשק IUnitOfWork המתארת את רכיבי המנשק –



**מאפיינים:** המנשק מכיל כמאפיינים מופעי מאגרי IRepository עבור כל הישויות העסקיות שהוא אחראי לסנכרן מול מסד הנתונים. המאגרים מסייעים למנשק בניהול המאגרים בזיכרון זמן ריצה של הישויות הקונקרטיות, כדי שהוא יוכל להתמקד במלאכת הסנכרון מול מסד הנתונים. בשלב זה המאגר היחידי שמנוהל על ידי המנשק הוא מאגר שירים. אם בהמשך יוחלט לשמור במסד הנתונים ישויות נוספות – למשל שמירת המנגינות שנוצרות בתהליך ההלחנה, לאחר יצירת טבלה מתאימה במסד הנתונים, מחלקת מודל ייעודית ומימוש מאגר ייעודי עבור מנגינות מולחנות, יש להוסיף את מנשק המאגר של המנגינות למנשק ה-UnitOfWork, כדי שיהיה זמין לקליינטים לאחזור ועדכון רשומות מנגינות, וכן שיהיה תחת הפיקוח של ה-UnitOfWork.

**מתודות**: להלן תקציר המתודות המוגדרות במנשק IUnitOfWork –

|  |  |
| --- | --- |
| **Description** | **Method** |
| Updates the underlying database, either synchronously or asynchronously, and returns the number of successfully updated records in the underlying data store. | CommitChanges |
| CommitChangesAsync |
| 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. | RegisterDirty |
| Releases managed resources that are no longer needed. | Dispose |

קוד המקור של המנשק מוגדר בקובץ IUnitOfWork.cs שניתן למצוא בקישור הבא –

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

### [מחלקת יחידת עבודה (EFUnitOfWork Class)](https://github.com/cwelt/Soloist/blob/master/DataAccess/UnitOfWork/EFUnitOfWork.cs)

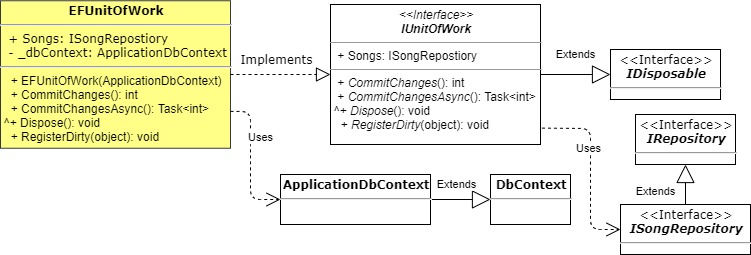
המחלקה EFUnitOfWork מממשת את המנשק IUnitOfWork של יחידת עבודה –

public class EFUnitOfWork : IUnitOfWork {…}

ה-EF בשם המחלקה הם ראשי-תיבות של [Entity Framework](https://docs.microsoft.com/en-us/ef/ef6/), חבילת ה-[ORM](https://he.wikipedia.org/wiki/%D7%9E%D7%99%D7%A4%D7%95%D7%99_%D7%90%D7%95%D7%91%D7%99%D7%99%D7%A7%D7%98%D7%99-%D7%A8%D7%9C%D7%A6%D7%99%D7%95%D7%A0%D7%99) מבית Microsoft. שם המחלקה מרמז על האופן שבו היא מממשת את המנשק [IUnitOfWork](#_מנשק_יחידת_עבודה): באמצעות הפונקציונאליות שכבר מסופקת "Out of the Box" ע"י חבילת ה-EF: החבילה מגדירה טיפוס DbContext המתפקד כמנהל ישויות תוך מימוש הדפוס הארכיטקטוני [Unit of Work](https://martinfowler.com/eaaCatalog/unitOfWork.html), ומחזיק אוסף ישויות גנריות מטיפוס DbSet, אשר כל אחת מהן מתפקדת למעשה כ-Repository עבור הישות מהטיפוס הקונקרטי של אותו DbSet. לפיכך, במקום לכתוב מאפס את כל הלוגיקה לסנכרון ה-Repositories מול ה-DB, ניתן בגוף מימוש מתודות המנשק פשוט לבצע האצלת אחריות (delegation) אל ישויות ה-EF, ובפרט אל ישות ה-DbContext. ספציפית בפרויקט זה, מאחר שמדובר ביישום אינטרנטי, נעשה שימוש בישות ה- ApplicationDbContextהיורשת מ- DbContext ומרחיבה את הפונקציונאליות בהתאמה ליישומי Web וניהול רשומות משתמש.

היתרון בשימוש במנשק ובמחלקה מגשרת זו עבור מימוש דפוס UnitOfWork במקום בשימוש ישיר ב-EF בקוד בצד הקליינטים (יתר שכבות המערכת) הוא האבסטרקציה: הקליינטים אינם תלויים ב-EF אלא רק במנשקים המופשטים של ה-IRepository וה-IUnitOfWork. התלות באבסטרקציה מאפשרת להחליף את המימוש הקיים של מחלקות אלו במחלקות אחרות, שמממשת את המנשקים ע"י חבילת ORM אחרת (למשל [NHibernate](https://nhibernate.info/)), או בכלל בשימוש ישיר של ספריית ה-ADO.NET. כל עוד קיים מימוש למנשקים IRepository ו- IUnitOfWork, תוך שימוש ב-Factory מתאים (או מנגנון Dependency Injection), לקליינטים שעובדים מול המנשקים בלבד (מול האבסטרקציה בלבד) המחלקות הקונקרטיות שמממשות את המנשקים שקופות לחלוטין.

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



המחלקה מגדירה בנאי יחיד המקבל כפרמטר מופע של יישות ApplicationDbContext, שהיא כאמור יורשת מ-DbContext ומספקת בפועל את התמיכה בפונקציונאליות הנדרשת לניהול ה-UnitOfWork באינטראקציה מול ה-DB. את המופע הזה היא שומרת במשתנה הפרטי \_dbContext.

המחלקה אינה מגדירה מאפיינים או מתודות נוספות מעבר לאלו שהיא מממשת מהמנשק IUnitOfWork.

קוד המקור של המחלקה מוגדר בקובץ EFUnitOfWork.cs שניתן למצוא בקישור הבא –

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

## [מיפוי בין אובייקטים לרשומות (EntityFramework ORM)](https://github.com/cwelt/Soloist/tree/master/DataAccess/EntityFramework)

העבודה מול בסיסי הנתונים במערכת אינה מבוצעת ישירות עם ה-API של ADO.NET, אלא עם Entity Framework – כלי ה-ORM (Object-Relational-Mapping) של Microsoft ליישומי .NET. כלי זה מממש טכניקת תכנות להמרה אוטומטית של נתונים מייצוגם כאובייקטים מורכבים בסביבת OOP לטבלאות שטוחות בבסיס נתונים רלציוני ולהפך. לאחר הגדרת הקונפיגורציה של המיפוי, העבודה מול בסיס הנתונים היא מול המודל הקונספטואלי בעולם האובייקטים, מבלי הצורך לרדת לעולם הפיזי של בסיסי נתונים, מה שחוסך התעסקות Low-level כגון ניהול Connection-ים ל-DB, בקיאות בשמות טבלאות, עמודות, המרת ייצוגים מטיפוסים ממערכות DBMS לטיפוסי .NET CLR, שאילתות SQL, וכד'. כל זאת מבוצע מאחורי הקלעים ע"י כלי ה-ORM – בהקשר של מערכת זו, הכלי שנבחר הוא Entity Framework.

התשתית והקונפיגורציה של המיפוי בין בסיס הנתונים הפיסי למודל הקונספטואלי בעולם האובייקטים (מחלקות ומנשקים) מרוכזת במרחב השמות הבא (או בתתי-מרחבים שלו) –

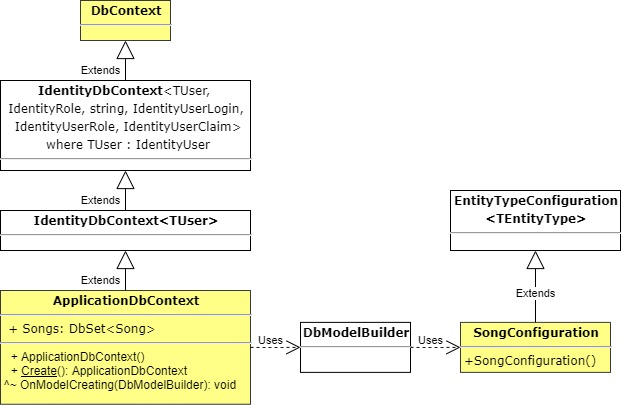
CW.Soloist.DataAccess.EntityFramework

קבצי קוד המקור המכילים רכיבים אלו נגישים בקישור הבא –

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

Entity Framework מאפשרת כמה גישות ודרכים להגדרה וניהול של קונפיגורציות המיפוי בין בסיס הנתונים הפיסי למודל הקונספטואלי בעולם התכנות מונחה עצמים. הגישה שנבחרה במסגרת פיתוח המערכת היא גישת "Code First", שאמנם דורשת יותר כתיבת קוד, ובפרט דורשת הגדרה ידנית מפורשת ללא מחוללים אוטומטיים מבוססי XML, אולם הגדרה מפורשת זו בקוד שמבוצעת ידנית מספקת רמת שליטה טובה יותר, ללא הסתמכות על תוצאות כלים אוטומטיים. הערה: זוהי גם הגישה היחידה הנתמכת ע"י Microsoft בגרסאות האחרונות של EF בפלטפורמת .NET Core: ב-Entity Framework Core ההגדרה מבוצעת אך ורק ב-Code First. באותה רוח, הגדרות המיפויים לא מבוצעות עם אנוטציות, אלא בקוד, באמצעות מחלקת DbModelBuilder המקבלת מופע של מחלקת קונפיגורציות ייעודית, כגון SongConfiguration, המגדירה את קונפיגורציות המיפוי של ישות מחלקת המודל Song אל מול הטבלה התאימה לה במסד הנתונים.

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



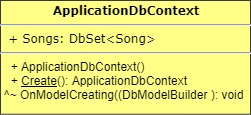
### [מחלקת קונטקסט מסד הנתונים (ApplicationDbContext Class)](https://github.com/cwelt/Soloist/blob/master/DataAccess/EntityFramework/ApplicationDbContext.cs)

המחלקה ApplicationDbContext מנהלת את האינטראקציה מול מסד הנתונים הפיסי –

public class ApplicationDbContext : IdentityDbContext<ApplicationUser> {…}

מחלקה זו יורשת מהמחלקה DbContext, המחלקה של Entity Framework שדואגת לממש 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 מופעי DbSet, ועוד. בהיררכיית עץ הירושה שבין DbContext לבין DbApplicationContext נמצאות מספר מחלקות גנריות של IdentityDbContext המרחיבות את הפונקציונאליות הבסיסית עם פונקציונאליות ייעודית לניהול משתמשים עם מימוש דפוס האבטחה AAA (Authentication, Authorization & Accounting) שמוכוון ליישומי Web המפותחים ב-ASP.NET MVC. הרחבות אלו חוסכות את הצורך בהגדרה מפורשת של טיפול ב-persistence של ישויות הקשורות לניהול משתמשים, כגון חשבונות משתמשים, קבוצות הרשאה (Roles) וכד'.

מחלקה זו מגדירה אוסף של מאגרי ישויות – DbSet, שמספקים גישה לניהול אוספי ישויות מאותו טיפוס במאגר זיכרון זמן ריצה, ומתודות לאתחול וקונפיגורציה של המיפויים –



להלן תקציר המתודות המוגדרות במחלקה, הראשונה – Create, היא מתודת Factory סטטית משמשת את סביבת ה-ASP.NET MVC הסטנדרטית לאתחול וקונפיגורציה ראשוניים של ה-EF, והשנייה – OnModelCreating, משמשת להגדרות מיפויים בקוד והיא נדרסת במחלקה זו לטובת הוספת מיפויים לישות של שיר וישויות פרטניות למערכת נוספות בעתיד במידת הצורך –

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

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 מהמחלקה הגנרית EntityTypeConfiguration, להחליף בה את הטיפוס הגנרי בטיפוס הקונקרטי של המודל הרלוונטי, ולהגדיר בבנאי את כל המיפויים, כפי שמבוצע לדוגמה במחלקה [SongConfiguration](#_מחלקת_מיפוי_ישות) (פירוט בהמשך).

קוד המקור של המחלקה מוגדר בקובץ ApplicationDbContext.cs שניתן למצוא בקישור הבא –

<https://github.com/cwelt/Soloist/blob/master/DataAccess/EntityFramework/ApplicationDbContext.cs>

### [מחלקת מיפוי ישות שיר לטבלה (SongConfiguration Class)](https://github.com/cwelt/Soloist/blob/master/DataAccess/EntityFramework/MappingConfigurations/SongConfiguration.cs)

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

internal class SongConfiguration : EntityTypeConfiguration<Song> {…}

מחלקה זו יורשת מהמחלקה הגנרית EntityTypeConfiguration<Song>. המחלקה מגדירה בגוף הבנאי שלה באמצעות מתודות יעודות מיפויים מול הטבלה המתאימה ב-DB – שם הטבלה, מיפוי מאפיינים (Properties) מול עמודות בטבלה, הגדרה איזה מאפיינים במחלקה אינם רלוונטיים למיפוי ושמירה ב-DB (כגון מאפייני ניווט), הגדרת אילוצים על העמודות (כגון הגבלת אורך מחרוזת בעמודת varchar), סימון המאפיינים שמרכיבים את המזהה החד-ערכי (מפתח) בטבלה, הגדרה האם מותרים ערכי null, וכן הגדרות נוספות רלוונטיות למיפוי בין הישות הקונספטואלית המיוצגת כמחלקה לבין הישות הפיסית ב-DB המיוצגת כרשומה שטוחה בטבלה בבסיס נתונים.

המתודות היעודות להגדרת הקונפיגורציות מחזירות כולן איזשהו אובייקט קונפיגורציה, כך שניתן לבצע הגדרה ב-"Fluent Api" ולשרשר קריאות קונפיגורציה בזו אחר זו בדומה לדפוס העיצוב Decorator, שכן כל מתודת קונפיגורציה מחזירה אובייקט קונפיגורציה מעודכן שאותו אפשר להמשיך ולקנפג.

קוד המקור של המחלקה מוגדר בקובץ SongConfiguration.cs שניתן למצוא בקישור הבא –

<https://github.com/cwelt/Soloist/blob/master/DataAccess/EntityFramework/MappingConfigurations/SongConfiguration.cs>

## [מיגרציות (Migrations)](https://github.com/cwelt/Soloist/tree/master/DataAccess/EntityFramework)

בשיטת ההגדרה של Entity Framework העובדת בהגדרות מפורשות בקוד (code first), שינויים בקוד המשפיעים על מסד הנתונים מייצרים מיגרציות (בהנחה שמיגרציות אופשרו בפרויקט עם הרצת הפקודה Enable-Migrations מתוך ה-Package Manager Console). מיגרציות אלו הן מחלקות היורשות מ-DbMigration, והן מכילות שתי מתודות: הראשונה – Up, האחראית לשדרג או להעלות גרסה במסד הנתונים, דהיינו להביא אותו למצב העדכני כפי שמיוצג במודל הקונספטואלי במחלקות בעולם התכנות מונחה עצמים, ואילו המתודה השנייה – Down, אחראית על ביצוע רגרסיה – מעין Rollback לגרסה קודמת, לפיכך מתודות אלו צריכות להכיל פעולות הופכיות זו לזו, למשל אם Up מכילה הוראה להקים טבלה, אזי Down צריכה להכיל הוראה למחוק טבלה זו (Drop). יש לקחת בחשבון שהגדרת פעולה הופכית יכולה לגרום לאיבוד נתונים אם לא נזהרים: למשל אם המצב הוא הפוך – ה-Up גורם למחיקת טבלה וה-Down גורם ליצירת הטבלה מחדש, יש לדאוג טרם מחיקת הטבלה לגבות את הנתונים (תוכן הטבלה) ולטעון אותם מחדש במסגרת פעולת ה-Down.

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

CW.Soloist.DataAccess.Migrations

בפועל הן שייכות כולן ל-Entity Framwork, ולכן טבעי יותר אולי היה לשמור אותם במרחב השמות של EntityFramewor, אולם כברירת מחדל הן נשמרות אוטומטית בספרייה ייעודית וזהו מרחב השמות ניתן להם בהתאמה. קוד המקור שלהן זמין בקישור הבא –

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

# שכבת התצוגה (PL)

שכבה התצוגה (PL – Presentation Layer) אחראית על האינטראקציה מול משתמשי הקצה – הגדרת ממשק משתמש גראפי (GUI), האזנה לקלט המשתמש ומתן מענה לבקשות שלו תוך אינטראקציה הן עם שכבת הגישה לנתונים לאחזור ועדכון רשומות ב-DB, והן שכבת הלוגיקה העסקית של האפליקציה לטובת צריכת שירותי ההלחנה שהמשתמש מעוניין בהן.

שכבה זו מוגדרת בשלמותה כספרייה בפרויקט Web ייעודי המהווה יחידת assembly עצמאית: WebApplication. בהתאם (וע"פ [מוסכמות השמות במערכת למרחבי שמות](#NamespaceConventions)), כלל מרחבי השמות (namespaces) השייכים לשכבה זו מושרשים תחת מרחב השמות: CW.Soloist.WebApplication.

פרויקט ה-Web כתוב בפלטפורמת ה-[[3]](#footnote-3)ASP.NET MVC. פלטפורמה זו מובנית על בסיס דפוס העיצוב הארכיטקטוני שעל שמה היא נקראת – **MVC**: **M**odel-**V**iew-**C**ontroller. ה--**Modelים** הם מחלקות POCO המייצגות את הישויות העסקיות, ה-**Views** הם החזית שהמשתמש רואה – בפרויקט זה אלו הם דפי HTML הנוצרים לאחר עיבוד קבצי cshtml המורכבים מתערובת של C#, HTML, Javascript ותחביר ייעודי של מנוע ה-Razor של ASP.NET MVC שמנתח קבצי cshtml ומייצר מהם קבצי HTML לאחר שיבוץ נתונים רלוונטיים מהמודלים, ולבסוף ה-**Controllers**, אלו הם בקרים המטפלים בבקשות משתמשי הקצה, מעבדים את הקלט, אוספים 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 נוספות שבהן ניתן להתערב ולדרוס או להרחיב את לוגיקת ברירת המחדל הקיימת בפלטפורמה. מסמך זה מתמקד רק ברכיבי הפיתוח ונקודות ההתערבות שבוצעו לטובת המערכת, ולא מפרט את הקוד הקיים של Microsoft שכבר מגיע מוכן 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 עץ התיקיות של הפרויקט –

|  |  |
| --- | --- |
| תוכן | תיקייה |
| נתונים השמורים במערכת הקבצים של שרת ה-Web, דוגמת לוגים, מסמכי PDF שזמינים להורדה מאתר האפליקציה, וקבצי ה-MIDI והטקסט של השירים. | App\_Data |
| מחלקות עם שגרות קונפיגורציה המופעלות עם אתחול האפליקציה, כגון רישום שירותים ומתודות/מחלקות שישמשו את האפליקציה בנקודות שונות. | App\_Start |
| משאבים סטטיים כמו קבצי css מספריית Bootstrap עבור עיצוב התצוגה. | Content |
| בקרים – המחלקות שמטפלות בבקשות השונות המגיעות מהמשתמש. | Controllers |
| מחלקות המגדירות פונקציונאליות פרטנית שמרחיבה/דורסת את הפונקציונאליות הסטנדרטית של .NET בנקודות עוגן מוגדרות מראש להתערבות, כגון הגדרת הלוגיקה לטיפול ב-Exception שלא נתפס ע"י סביבת זמן הריצה. | Filters |
| סט פונטים מוכן שמגיעים אוטומטית עם הקמת פרויקט ASP.NET MVC. | fonts |
| סט סקריפטים מוכן ב-javascript מספריות JQuery ו-Bootstrap שמגיע אוטומטית עם הקמת פרויקט ASP.NET MVC. | Scripts |
| מחלקות ולידציה שניתן להפעיל על שדות בטפסי HTML באמצעות אנוטציות. | Validations |
| מחלקות [DTO (Data Transfer Objects)](https://en.wikipedia.org/wiki/Data_transfer_object) ייעודיות ל-Views. אלו הם למעשה פשוט מודלים שנוצרו בהתאמה אישית ל-Views. | ViewModels |
| דפי Razor (cshtml) המתורגמים לדפי HTML מועשרים בנתונים. | Views |
| משאבים וקבצי קופיגורציה גלובאליים כגון packages.config – קובץ ה-XML לניהול תלויות מספריות חיצוניות (בדומה לקובץ POM ביישומי Java שעוברים הידור עם Maven), Global.asax לרישום גלובאלי של מחלקות לטיפול באירועים אפליקטיביים, ו- favicon.ico – האייקון המוצג בדפדפן עבור דף האפליקציה. | / (root) |

הסעיפים הבאים מפרטים את כל הפיתוחים והרחבות שבוצעו בפרויקט ה-ASP.NET MVC הסטנדרטי לטובת המערכת.

## תצורת התוכן האפליקטיבי על השרת (App\_Data)

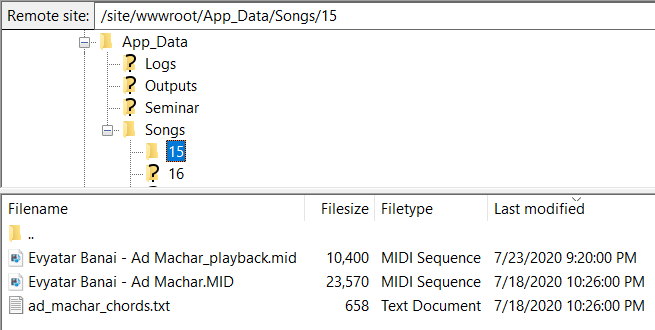
התיקייה App\_Data השמורה על השרת מכילה מסמכים וקבצים שמשמשים את האפליקציה. תיקייה זו מכילה 4 תתי-תיקיות: Logs, Songs, Outputs ו-Seminar. סעיף זה מפרט את תוכן כל אחת מתתי-תיקיות אלו.

### לוגים (Logs)

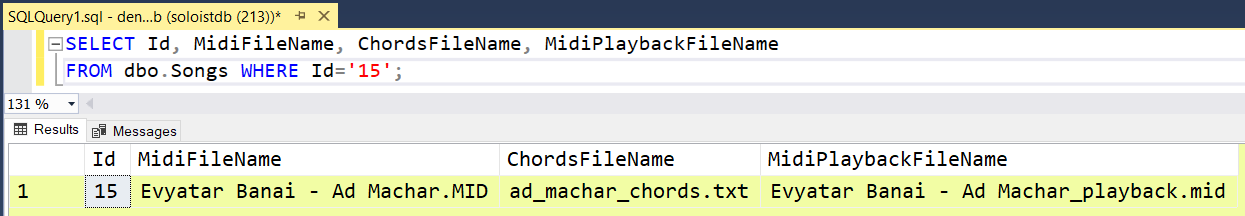
התיקייה Logs מכילה קבצי לוגים שהמערכת רושמת לתוכם לצורכי בקרה: למשל – ErrorLog.txt מכיל פירוט שגיאות Exception בלתי צפויים שהתקבלו, ו- RequestLog.txt מתעד פרטי בקשות של משתמשים באתר.

### שירים (Songs)

התיקייה Songs מכילה את קבצי השירים שפרטיהם מאוחסנים ב-DB. הפרטים של השירים מנוהלים ב-DB, אולם הקבצים עצמם שמורים על השרת – אלו הם קבצי מידי וקבצי טקסט שכמעט ואינם צורכים זיכרון.הקבצים של כל שיר נשמרים בתת-תיקייה משלו בתיקיה Songs. כאשר שיר חדש נשמר ב-DB, הוא מקבל מספר מזהה חד-ערכי: Id. המערכת יוצרת עבורו תיקייה חדשה בתיקייה Songs תחת Id זה, ומרכזת שם את הקבצים הרלוונטיים לשיר. מצ"ב לדוגמה תצלום מהתוכנה FileZila המשמשת כ-FTP Client לחיבור מרוחק לשם העברת קבצים בין תחנת עבודה לשרת ה-Web. בתצלום המסך ניתן לראות בחלק העליון את מבנה תיקיית Songs על השרת כפי שתואר לעיל, ובחלק התחתון את קבצי השיר שה-Id שלו הוא 15 –

****

ב-DB, ברשומות השיר רשומים שמות הקבצים השונים (קובץ אקורדים, קובץ MIDI וקובץ פלייבק) שבתוך התיקייה 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 –



### מנגינות (Outputs)

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

### סמינר (Seminar)

תיקיית Seminar מכילה מסמך עבודת סמינר בנושא הלחנה עם אלגוריתמים גנטיים. עבודה זו מכילה את הבסיס התאורטי למימוש האלגוריתם המרכזי במערכת. המסמך זמין להורדה באתר האפליקציה ולכן שמור ישירות על השרת לטובת גישה מהירה.

## קונפיגורציות אתחול אפליקציה (App\_Start)

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

### הגדרת חבילות (BundleConfig)

המחלקה BundleConfig מכילה הגדרות לאריזה של קובצי javascript ו-css לחבילות. לטובת עדכון עיצוב ברירת המחדל של bootstrap המוגדרת ב-ASP.NET MVC, קובץ ה-bootstrap.css שמתווסף כברירת מחדל ל-bundle הוחלף ב-template אחר: bootstrap-darkly.css. קובץ זה הורד מאתר bootstrap ונשמר בתיקיית המשאבים הסטטיים Content בתת-תיקייה Themes.

### רישום מחלקות פילטר (FilterConfig)

המחלקהFilterConfig מכילה רישום הגדרת מחלקות המגדירות לוגיקה פרטנית מותאמת אישית שמחליפה/מרחיבה את לוגיקת ברירת המחדל הסטנדרטית שמסופקת ע"י פלטפורמת ה-ASP.NET MVC. במחלקה זו מבוצע רק רישום מחלקות אלו, ובגוף המחלקות עצמן נרשמות נקודות העוגן שבהן הן מעוניינות להתערב – למשל התערבות לפני ביצוע מתודה ע"י בקר. מחלקות אלו נקראות מחלקות פילטר, שכן בהתערבויות התעבורה עוברת דרכם כמו פילטר, והן יכולות להחליט מה לעשות עם התעבורה והאם לסנן לחסום/לאפשר וכד' בדומה ל-Firewall. ניתן למשל להגדיר מחלקת פילטר שבודקת האם התעבורה מגיעה ממנוע חיפוש (Web crawler engine) ובמידה וכן היא תסנן את הבקשה ותחזיר שגיאה כגון 404 (Not Found). להלן מחלקות הפילטר שהמערכת מגדירה בעצמה ורושמת כמחלקות פילטר במסגרת ה-FilterConfig. המחלקות עצמן מוגדרות בתיקייה Filters –

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

### הגדרת שירותי זהויות (IdentityConfig)

הקובץ IdentityConfig.cs מרכז מחלקות שירות שונות המסייעות בתהליך הזיהוי ואימות של משתמשים, כגון מחלקות לקביעת מדיניות אבטחה לגבי הסיסמה ושירותי שליחת מייל ו-sms למשתמש ששכח סיסמא ו/או אמצעי זיהוי נוסף עבור two-factor authentication. מחלקות אלו מגיעות אוטומטית עם ה-ASP.NET MVC אם בוחרים באפשרות של ניהול זהויות בעת הקמת הפרויקט, עם זאת, המחלקות לא מכילות פונקציונאליות שלמה. בפרט – עבור שירותי שליחת מייל ו-sms, פלטפורמת ה-ASP.NET MVC מספקת מחלקות שירות עם מתודות ריקות המתפקדות כ-"placeholders", נקודות עוגן שאותן יש לממש עצמאית.

המערכת מממשת בשלב זה את שירות שליחת המייל, שירות זה מוגדר במחלקה EmailService במסגרת המתודה SendAsync. הלוגיקה של שליחת המייל מוגדרת במתודה נפרדת שנקראת מתוך מתודת השירות SendAsync. כרגע מוגדרים שני מימושים שונים, האחד ממומש באמצעות שירותי ה-SMTP הבסיסיים המסופקים ע"י .NET: SendEmailAsyncWithSMTP, והשני ממומש תוך שימוש בשירותי ספרייה חיצונית SendGrid: SendEmailAsyncWithSendGridService. נכון לעכשיו המתודה SendAsync מוגדרת להפעיל את מתודת שליחת המייל במימוש של SendGrid.

### הגדרת שירותי הרשאה (Startup.Auth)

המחלקה Startup מגדירה שירותי הרשאה שונים, ביניהם האפשרות לבצע אימות חשבונות מול אפליקציות צד ג' באמצעות פרוטוקול [OAuth 2.0](https://auth0.com/docs/protocols/protocol-oauth2?utm_source=emea-google-dg&utm_medium=cpc&utm_campaign=Branded&utm_content=dynamicRT1&utm_term=&gclid=Cj0KCQjw5eX7BRDQARIsAMhYLP_4WJFKc1r9dCb3Djuf-DqfeLJdHM1UjE2OyF0zgeHTWyuZELT_z4EaAlrdEALw_wcB). בשלב זה מוגדרת האפשרות לבצע אימות שכזה מול חשבונות Google, ע"י קריאה למתודה UseGoogleAuthentication על מופע ה- IAppBuilder תוך אספקה פרטי הזדהות של האפליקציה שהתקבלו ברישום מול Google. פרטים אלו מאוחסנים בקובץ קונפיגורציות ייעודי על השרת שאינו חשוף באופן ציבורי – AppSensitiveSettings.config.

## בקרים (Controllers)

בתיקיית Controllers מרוכזות כלל מחלקות הבקרים. בהתאם, הן מוגדרות ב-namespace הבא –

CW.Soloist.WebApplication.Controllers

קוד המקור של כלל מחלקות אלו זמין בקישור הבא –

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

סעיף זה מפרט את מחלקות הבקרים השונות. בהתאם לקונבנציות של ASP.NET MVC, כל בקר מוגדר במחלקה נפרדת משלו היורשת ממחלקת האב האבסטרקטית [Controller](https://docs.microsoft.com/en-us/dotnet/api/system.web.mvc.controller?view=aspnet-mvc-5.2), ושם המחלקה שלו היא שם הבקר בתוספת הסיומת Controller. למשל – SongsController הוא שם מחלקת בקר השירים.

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



להלן תקציר הבקרים השונים –

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

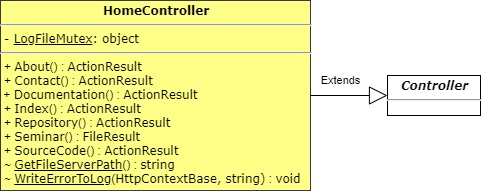
### בקר דף הבית (HomeController)

המחלקה HomeController מטפלת בבקשות המגיעות מתוך הבית של אתר האפליקציה וכן בבקשות שירות כלליות שרלוונטיות לכלל הבקרים כגון קבלת נתיב לספריית השירים –

public class HomeController : Controller {…}

#### דיאגרמת מחלקה

להלן דיאגרמת מחלקה של בקר דף הבית HomeController –



#### מאפיינים

המחלקה מגדירה שדה יחיד – מנעול למניעה הדדית של גישה לקובצי לוגים במקביל –

private static readonly object LogFileMutex = new object();

#### מתודות

להלן חתימות המתודות הציבוריות המחזירות עם תקציר תיעוד בסיסי שלהן –

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

ע"פ קונבנציית ברירת המחדל, מתודות אלו נקראות ע"י נתב האפליקציה בעת גישה לנתיב של דף הבית (שם הבקר) עם שם המתודה (שם ה-Action). למשל: ה-URL הבא מנווט 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 המחזירות דפים כתגובה לגלישה של משתמש ל-URL מתאים, בקר דף הבית מגדיר שם שני מתודות שירות כלליות לשימוש פנימי בין הבקרים: הראשונה מחזירה את הנתיב לתיקייה על השרת שבה שמורים הקבצים (App\_Data), והשנייה משמשת את הבקרים לרישום שגיאות בקובץ לוג השגיאות –

internal static string GetFileServerPath(HttpServerUtilityBase server = null){…}

internal static void WriteErrorToLog(HttpContextBase context, string errorMessage) {…}

### בקר שירים (SongsController)

המחלקה SongsController אחראית לטפל בבקשות לניהול מאגר השירים באפליקציה –

public class SongsController : Controller {…}

במסגרת האחריות שלה המחלקה מספקת תמיכה בפעולות CRUD (יצירה, אחזור, עדכון ומחיקה) של משאבי שירים במערכת, בכפוף לבדיקת הרשאות המשתמש. היא אחראית בין היתר על מסך אינדקס השירים. להלן קישור למסך זה על גבי השרת הנוכחי המארח את האפליקציה –

<https://soloist.gear.host/Songs>

#### דיאגרמת מחלקה

להלן דיאגרמת מחלקה של בקר השירים SongsController –



#### מאפיינים

המחלקה מגדירה שני מאפיינים – ידית גישה לאינטראקציה מול ה-DB, ונתיב לתיקייה על השרת שבה מנוהל מאגר קבצי השירים –

private IUnitOfWork \_databaseGateway;

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

#### בנאי

המחלקה מגדירה בנאי המציין את התלות שלו במופע מנהל ישויות DB ביחידת עבודה –

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

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

#### מתודות

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 –

|  |  |
| --- | --- |
| Description | Method |
| 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. | Create |
| 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. | Delete |
| DeleteConfirmed |
| Handles a HTTP GET request for displaying a detailed screen for a specific song. | Details |
| Releases managed resources that are no longer required. | Dispose |
| Downloads a requested song file to the end user's computer. | DownloadFile |
| 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. | Edit |
| Gets the path of the given song's resources on the hosting file server. | GetSongPath |
| Gets and displays song list that user is authorized to see. | Index |
| Checks if a user is authorized for the given song and activity. | IsUserAuthorized |

### בקר הלחנה (CompositionController)

המחלקה CompositionController אחראית לטפל בבקשות להלחנת מנגינות חדשות –

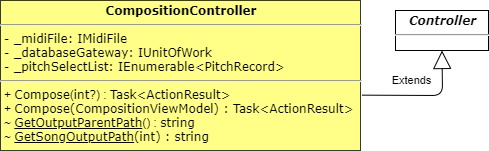
public class CompositionController : Controller {…}

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

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

#### דיאגרמת מחלקה

להלן דיאגרמת מחלקה של בקר ההלחנה CompositionController –



#### שדות

המחלקה מגדירה שלושה שדות –קובץ MIDI של פלט המנגינה שתיווצר בתהליך ההלחנה, ידית גישה לאינטראקציה מול ה-DB, ורצף פנימי של צלילים לטובת מנשק המשתמש –

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

#### בנאי

המחלקה מגדירה בנאי המציין את התלות שלו במופע מנהל ישויות 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 התלויות בקונפיגורציית ה-IoC (Inversion Of Control) בקובץ Global.asax. באופן זה, ניתן להחליף את מימוש ה-DB לחלוטין באופן שקוף לבקר ביישום ה-Web, כל עוד המימוש החדש מקיים את דרישות המנשק IUnitOfWork.

#### מתודות

להלן חתימות המתודות במחלקה –

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 תיעוד בסיסי של המתודות שלעיל –

|  |  |
| --- | --- |
| Description | Method |
| 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. | Compose |
| 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. | GetOutputParentPath |
| GetSongOutputPath |

#### מחלקת רשומת גובה צליל (PitchRecord Class)

המחלקה PitchRecord היא מחלקה פרטית פנימית המקוננת במחלקת בקר השירים, שמשמשת להגדרת מבנה רשומת עזר מותאמת לממשק המשתמש –

private class PitchRecord {

public NotePitch Pitch { get; set; }

public string Description { get; set; } }

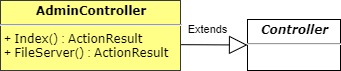
### בקר ניהול (AdminController)

המחלקה AdminController אחראית לטפל בבקשות מנהלן מערכת לניהול האפליקציה –

public class AdminController : Controller {…}

הבקר חושף את ממשק משתמש ההנהלה רק עבור משתמשים מורשים. ממשק זה זמין בקישור הבא (על השרת הנוכחי המארח את האפליקציה): <https://soloist.gear.host/Admin>

להלן דיאגרמת מחלקה עבור בקר הניהול –



בשלב זה ממשק הניהול מספק רק שירות בסיסי של הפנייה לשרת הקבצים שעל השרת. שירות\*י ניהול נוספים כגון ניהול רשומות משתמשים מבוצעים בשלב זה ע"י עדכון ישיר של הרשומות הרלוונטיות מתוך מערכות DBMS. עם זאת, הקמת ממשק הניהול מהווה תשתית להרחבות עתידיות. אלו מתודות המחלקה –

public ActionResult Index() {…} // Displays administration page

public ActionResult FileServer() {…} // Redirects to file server

### בקרי ניהול חשבונות משתמשים

המחלקותAccountController ו- ManageControllerהמנהלות את חשבונות המשתמשים מסופקות ע"י הפלטפורמה הסטנדרטית של ASP.NET MVC ומתועדות באתרי Microsoft. העדכון היחידי שבוצע מחלקות אלו הוא הפנייה לקונטקסט ה-Entity Framework משכבת הגישה לנתונים במקום זה שהגיע כברירת מחדל בפרויקט ה-Web שהוסר למניעת כפילות קונטקסטים.

## פילטרים (Filters)

פילטרים הם מחלקות המגדירות לוגיקה לביצוע בנקודות עוגן מוגדרות מראש ש-Microsoft אפשרו בפלטפורמת ה-ASP.NET MVC. הרעיון דומה לדפוס העיצוב Template Method: הפלטפורמה מגדירה את המנגנון ומספקת מימוש ברירת מחדל בנקודות עוגן (לעיתים זהו מימוש ריק), ומאפשרת לנו המפתחים לדרוס את ברירת המחדל ולספק מימושים שלנו בנקודות ההתערבות.

פלטפורמת ה-ASP.NET MVC מספקת כבר ארבעה סוגי פילטרים OOB (Out Of the Box), העונים לצרכים שונים ומספקים נקודות התערבות שונות. להלן ארבעת סוגי פילטרים אלו והמנשק שעליהם לממש –

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

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

כלל הפילטרים יורשים ממחלקת האב האבסטרקטית [FilterAttribute](https://docs.microsoft.com/en-us/dotnet/api/system.web.mvc.filterattribute?view=aspnet-mvc-5.2). ככאלו, ניתן להפעיל אותם באופן מקומי על בקרים/מתודות ע"י סימונם באנוטציה מתאימה. למשל, פעולת הבקר של העלאת שיר חדש למערכת סומנה באנוטציה Authorize המנחה את פילטר ה- Authorization לחסום פניות ממשתמשים לא מזוהים/רשומים –

[Authorize]

public ActionResult Create() {…} // Upload new song

במקום שימוש באנוטציות לתיחום מקומי של הפילטרים, ניתן לבצע רישום גלובאלי ברמת האפליקציה כולה מתוך מתודת הרישום הנקרית מהקובץ Global.asax: המתודה הסטטית RegisterGlobalFilters של המחלקה FilterConfig השמורה בתיקייה App\_Start שנסקרה לעיל.

באותם מקרים בהם יש צורך לדרוס ו/או להרחיב את לוגיקת ברירת המחדל המסופקת ע"י הפלטפורמה של .NET, עלינו להגדיר פילטרים מותאמים אישית לצרכים שלנו (Custom Filters). סעיף זה מתאר את מחלקות הפילטר המותאמות (Custom Filters) שנכתבו לטובת המערכת. מחלקות אלו מגדירות לוגיקה פרטנית שניתן לשבץ בנקודות עוגן מוגדרות מראש ש-Microsoft אפשרו בפלטפורמת ה-ASP.NET MVC. כלל המחלקות הללו שמורות בפרויקט בתיקייה Filters ובהתאם מוגדרות ב-namespace הבא ו/או בתתי-מרחבים שלו –

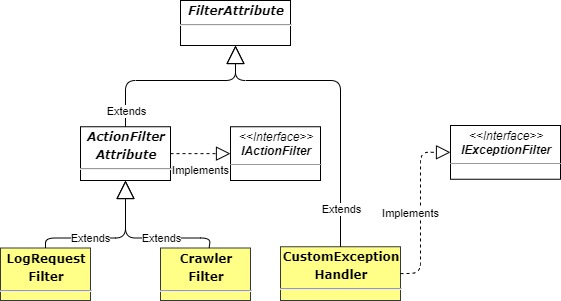
CW.Soloist.WebApplication.Filters

קוד המקור של כלל מחלקות אלו זמין בקישור הבא –

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

המערכת דורסת/מרחיבה את ברירת המחדל עם פילטרים מותאמים אישית משני סוגים: פילטרים מסוג Action להתערבות סביב פעולות של בקרים מסוימים (פירוט בהמשך), ופילטר מסוג Exception לדריסת טיפול ברירת המחדל בשגיאות חריגה שלא נתפסו ע"י האפליקציה.

להלן דיאגרמת מחלקה עם מחלקות הפילטר המותאמות שנכתבו לטובת המערכת (אלו המודגשות בצבע) והקשר שלהן למנשקים שהן מממשות ומחלקת-האב האבסטרקטית שהן יורשות ממנה –



### פילטר טיפול בחריגות (CustomExceptionHandler)

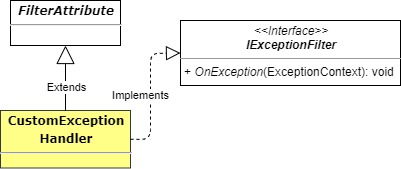
המחלקה CustomExceptionHandler מטפלת בשגיאות Exceptions שלא נתפסו ע"י האפליקציה –

public class CustomExceptionHandler : FilterAttribute, IExceptionFilter {…}

מחלקת פילטר זו מממשת את המתודה OnException של המנשק IExceptionFilter לטובת החזרת דף HTML מעוצב במקרה של חריגה שלא נתפסה, במקום דף שגיאת ברירת המחדל ש-.NET מחזירים. הדף המוחזר נשמר בקונטקסט ה מועבר כפרמטר. להלן חתימת המתודה –

public void OnException(ExceptionContext filterContext) {…}

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

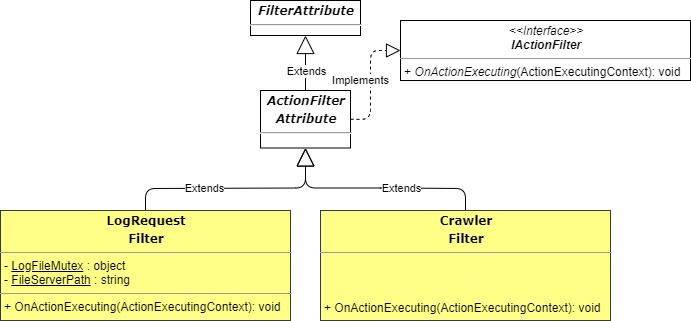


### פילטר טיפול בבקשות (ActionFilters)

בתיקייה ActionFilters תחת תיקיית Filters בפרויקט מוגדרות מחלקות פילטר מסוג Action המגדירות טיפול פרטני סביב פעולות בקרים, המאפשרות להתערב הן לפני והן אחרי ביצוע הפעולה ע"י הבקר, וכן לפני החזרת התוצאה מפעולה זו ולאחר החזרת התוצאה.

בשלב מומשו שני מחלקות Action Filter עבור המערכת: הראשונה – CrawlerFilter, האחראית על סינון גישה של בוטים ומנועי חיפוש למתודות 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 שתיהן מממשות מתודה יחידה OnActionExecuting מהמנשק IActionFilter.

להלן דיאגרמת מחלקה של מחלקות פילטר אלו המתארת את הקשר שלהן לרכיבי הפילטר הסטנדרטיים בפלטפורמת ה-ASP.NET MVC עם המתודה הרלוונטית שהן מממשות –



#### LogRequestFilter

public class LogRequestFilter : ActionFilterAttribute {…}

מחלקה זו מוגדרת לפעול באופן גלובאלי באפליקציה באמצעות רישום במתודה הסטטית RegisterGlobalFilters של המחלקה FilterConfig כפי שתואר לעיל. היא רושמת פרטים כלליים על כל בקשת HTTP המועברת לבקרים ושומרת פרטים אלו בקובץ לוג על השרת ב-App\_Data בתיקייה Logs. לטובת ניהול הרישום בלוגים המחלקה מגדירה שני שדות: FileServerPath – מחרוזת המכילה את הנתיב לתיקיית הקבצים של השרת (נכון לעכשיו זוהי תיקיית App\_Data), ו- LogFileMutex – מנעול למניעה גישה במקביל של שני Thread-ים או יותר בעת כתיבה בלוג.

#### CrawlerFilter

public class CrawlerFilter: ActionFilterAttribute {…}

מחלקה זו מוגדרת באופן מקומי על מתודות ייעודיות שעבורן המערכת חוסמת גישה לבוטים ומנועי חיפוש. לשם כך המתודות הייעודיות מסומנות באנוטציה מתאימה: [CrawlerFilter]. למשל – מתודת Contact בבקר דף הבית שמציגה למשתמש את פרטי יצירת הקשר של מנהלן האתר חסומה בפני גישת מנועי חיפוש ובוטים על מנת שלא יאחזרו מידע השמור השם ויאחסנו אותו במהלכים אוטומטיים –

[CrawlerFilter]

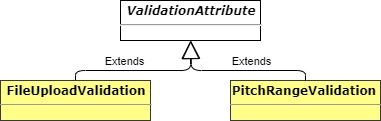
public ActionResult Contact() => View();

## מחלקות ולידציה (Validations)

פלטפורמת ה-ASP.NET MVC מגדירה מחלקות ולידציה לבדיקות תקינות, שאותן ניתן להפעיל של שדות בטופס HTML שמשתמשים שולחים לשרת ע"י סימון השדות המבוקשים לבדיקה באנוטציות מתאימות. כלל המחלקות הללו יורשות ממחלקת-האב האבסטרקטית ValidationAttribute.

הפלטפורמה מגדירה כבר סט בדיקות תקינות מוכנות מראש, כגון בדיקת תקינות כתובת מייל, בדיקה שערך נומרי נמצא בטווח מסוים, בדיקה ששדה אינו ריק ועוד. כמו כן, ניתן להגדיר מחלקות ולידציה מותאמות אישית שמגדירות בדיקות תקינות ע"פ הלוגיקה העסקית הנדרשת לאפליקציה. לאחר הגדרת מחלקות אלו, ניתן לסמן את השדות המבוקשים באנוטציות המתאימות ובדיקות 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 שלהן למחלקת-האב האבסטרקטית –



שתי מחלקות אלו מממשות את המתודה האבסטרקטית IsValid המקבלת כקלט אובייקט וקונטקסט רלוונטי, בודקת את ערך האובייקט ע"פ הלוגיקה המבוקשת בהתאם לתוכן הקונטקסט ומחזירה ValidationResultלמנגנון הבדיקות של .NET שבודק את התוצאה וממשיך את העיבוד בהתאם. להלן פירוט מימוש שתי מחלקות הולידציה.

### בדיקת תקינות קבצים (FileUploadValidation)

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

internal class FileUploadValidation : ValidationAttribute {…}

הבדיקה מבוצעת ע"י מימוש המתודה האבסטרקטית IsValid שקוראת בתורה למתודות עזר פנימיות האחראיות על בדיקות שונות –

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

בדיקת תקינות זו מופעלת ע"י סימון שדות שמשמשים להעלות קבצים למערכת עם אנוטציה מתאימה של [FileUploadValidation]. למשל שדה של קובץ ה-MIDI ב-SongViewModel, המודל שמשמש את טופס העלאת שיר חדש –

[FileUploadValidation]

public HttpPostedFileBase MidiFileHandler { get; set; }

להלן מתודות הולידציה הפנימיות שהמתודה IsValid נעזרת 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 (ווידוא שהקובץ אינו חורג מהגודל המקסימלי המוגדר) וסוג הקובץ (לוודא שרק קובצי MIDI וטקסט מועלים ולא איזה סקריפט זדוני). המתודה IsChordsFileValid מבצעת בודקת שקובץ האקורדים בפורמט המתאים שהוגדר, והמתודה IsMidiFileValid בודקת שקובץ ה-MIDI תקין. שתי המתודות האחרונות נעזרות בבדיקות שכבר הוגדרו במחלקה [CompositionContext](#_מחלקת_קונטקסט_ההלחנה).

להלן השדות המוגדרים במחלקה, המסייעים בשימוש חוזר במשתנים במקום הגדרתם מחדש בכל מתודת בדיקה בנפרד. כמו כן מוגדר משתנה סטטי קבוע להגבלת הגודל המקסימאלי על קובץ המועלה למערכת (נכון לעכשיו מוגדר כמגה בייט) –

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;

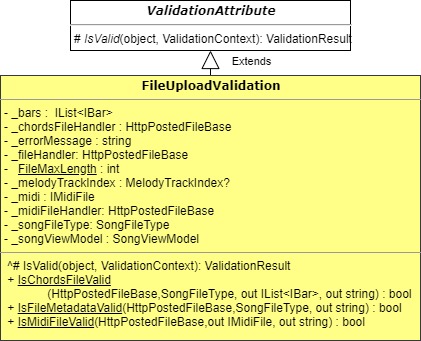
private HttpPostedFileBase \_fileHandler;

private HttpPostedFileBase \_midiFileHandler;

private HttpPostedFileBase \_chordsFileHandler;

private MelodyTrackIndex? \_melodyTrackIndex;

להלן דיאגרמת מחלקה של FileUploadValidation –



### בדיקת תקינות טווח צלילים (PitchRangeValidation)

המחלקה PitchRangeValidation מממשת בדיקות תקינות עבור טווח הצלילים שהמשתמש בוחר בטופס הלחנת שיר –

public class PitchRangeValidation : ValidationAttribute {…}

הבדיקה מבוצעת ע"י מימוש המתודה האבסטרקטית IsValid שקוראת בתורה למתודה הסטטית IsPitchRangeValid של המחלקה [CompositionContext](#_מחלקת_קונטקסט_ההלחנה) שמנהלת בפועל את לוגיקת הבדיקה –

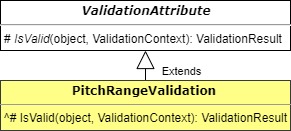
protected override ValidationResult IsValid(object value, ValidationContext context){…}

בדיקת תקינות זו מופעלת ע"י סימון השדה/שדות שקובעים את טווח את הצלילים ב-ViewModel שנשלח לטופס ההלחנה (CompositionViewModel) עם אנוטציה מתאימה של [PitchRangeValidation] –

[PitchRangeValidation]

public NotePitch MaxPitch { get; set; }

להלן דיאגרמת מחלקה של PitchRangeValidation –



## מודלים לטפסים (ViewModels)

ViewModels הם מחלקות POCO[[4]](#footnote-4) המהוות אוסף שדות (בדומה ל-struct בשפת C) ייעודי לטובת איזשהו View – מסך/טופס במנשק המשתמש. בהקשר של מערכת זו, תוצר ה-View הוא דף HTML. מנגנון מנוע ה-Views של ASP.NET MVC משבץ נתונים בתבניות ה-View המוגדרות. נתונים אלו משובצים מתוך מודל – ViewModel המסופק ל-View כקלט. להבדיל מהמודלים הנידונים ב[שכבת הגישה לנתונים](#_מודלים_(Domain_Models)), ה-ViewModel אינם רלוונטיים ל-Persistence. הם נוצרו במיוחד לשירות ה-Views במנשק המשתמש ולשם כך מאגדים שדות נוספים לבקרת הרשאות והעשרת ממשק המשתמש. סעיף זה מפרט את הגדרות ה-ViewModel השונים. כלל ה-ViewModel מרוכזים בפרויקט ה-Web בתיקייה ViewModels.

### מודל-תצוגה עבור שיר (SongViewModel Class)

המחלקה SongViewModel מגדירה מודל לריכוז הנתונים הרלוונטיים למסכים טפסים להצגה ועיבוד של ישויות שירים: העלאת שיר חדש, הצגת שיר קיים, עדכון שיר, מחיקת שיר והצגת רשימת השירים הקיימים. להלן המאפיינים/שדות המוגדרים במודל 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 באנוטציות שונות לבדיקות תקינות והגדרת תיאור ידידותי יותר למשתמש.

המחלקה מגדירה גם שני בנאים: בנאי ריק, ובנאי המשתמש כמעין בנאי העתקה, המקבל כקלט מופע של Song, מחלקת מודל שיר, ובונה ממנה את ה-ViewModel –

public SongViewModel() { }

public SongViewModel(Song song) { … }

### מודל-תצוגה עבור הלחנת מנגינה (CompositionModel Class)

המחלקה CompositionViewModel מגדירה מודל לריכוז הנתונים הרלוונטיים לטופס הלחנת שיר. להלן המאפיינים המוגדרים במודל 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 { … } } //Sums up all proportional weights

המאפיינים המאותחלים עם ערך נומרי כברירת מחדל מציינים את המשקל היחסי שיינתן למדד המיוצג ע"י אותו מאפיין בשערוך האלגוריתם הגנטי: למשל, המאפיין SmoothMovement מייצג את המדד של מעבר חלק בין צלילים שכנים שנאמד ע"י המתודה EvaluateSmoothMovement המתוארת [במודול השערוך](#_מודול_שערוך_(FitnessEvaluators.cs)). מדד זה מקבל כברירת מחדל משקל של 15% בשערוך הכללי. סך המשקלים צריך להסתכם ל-100.

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

### מודל-תצוגה עבור ניהול חשבונות משתמשים

פלטפורמת ה-ASP.NET MVC מגדירה שני קבצי ViewModel המרכזים מחלקות מודל לטובת המסכים של ניהול חשבונות משתמשים – הזדהות משתמשים, שחזור סיסמא וכד': AccountViewModels.cs ו-ManageViewModels.cs. תיעוד על מחלקות ViewModel אלו ניתן למצוא באתר Microsoft.

## מסכים וטפסים **(Views)**

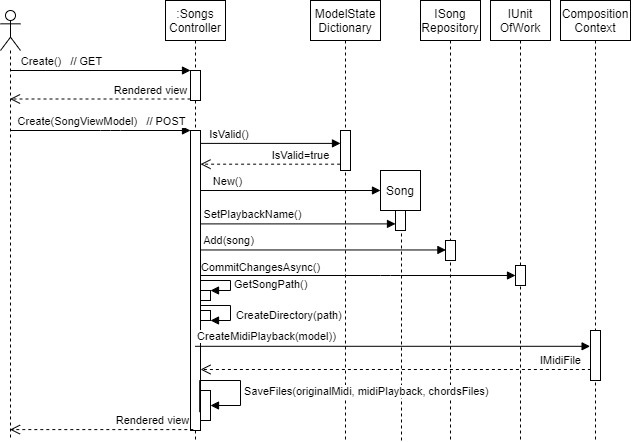
ה-Views בארכיטקטורת MVC הם המסכים והטפסים שמוצגים למשתמשים. עבור יישום ה-Web של המערכת, אלו הם למעשה דפי HTML. ע"פ הגדרת ברירת המחדל בארכיטקטורה של ASP.NET MVC, כלל ה-Views מרוכזים בתיקיית Views בפרויקט ה-Web, תחת המוסכמה הבאה – כל ה-Views (מסכים, טפסים – דפי HTML) שנשלחים אל המשתמש מתוך בקר מסוים, מרוכזים בתיקיית Views תחת תת-תיקיה ששמה הוא שם הבקר (ללא הסיומת Controller), ושם ברירת המחדל של קובץ ה-View (קובץ ה-Razor עם סיומת cshtml שמנגנון ה-ASP.NET MVC ממיר ל-HTML) הוא שם המתודה של הבקר שמחזירה View זה. למשל, הקובץ Create.cshtml בתיקיית Songs שתחת תיקיית Views הוא התבנית המשמשת את מנוע ה-Views של ASP.NET MVC לבניית דף HTML כאשר משתמשים פונים עם URL מתאים לבקר השירים למתודה Create (מנקודת המבט של המשתמש, כאשר מבוצע ניווט למסך העלאת שיר חדש באפליקציה).

בשל הקשר ההדוק בין ה-Views לבין מתודות הבקרים שמנווטות אליהן, פשר ומהות ה-Views השונים הם כפי שמתוארים במתודות הרלוונטיות בסעיף [הבקרים](#_בקרים_(Controllers)). למען הסדר הטוב, להלן תקציר תיקיות ה-Views השונים המוגדרים במערכת (התוכן עצמו של תיקיות אלו הוא ה- Views המתאימים למתודות שהתיעוד שלהן מפורט בסעיף ה-Controllers) –

|  |  |
| --- | --- |
| Description | Views Sub-Folder |
| ASP.NET MVC Predefined views for assisting the management of user AAA (Authentication, Authorization and Accounting). | Account |
| Manage |
| HTML form for composition a new melody for an existing song which is already stored in the system. | Composition |
| Views that are returned from actions that are managed by the home page controller such as about page, contact page, etc. | Home |
| 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. | Shared |
| Screen and forms that are managed by the Song controller for uploading, querying and updating songs. | Songs |

## אינטראקציה בין רכיבי ה-MVC

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



# דפוסי עיצוב

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

## דפוסי יצירה (Creational Patterns)

### דפוס מפעל (Factory Pattern)

אחד מעקרונות הפיתוח המובילים שהוטמעו בפיתוח המערכת הוא –

"***Program to an Interface, not an Implementation***"

בהתאם, כמעט לכל מחלקה פונקציונאלית במערכת מוגדר מנשק צל שמחצין ומגדיר באופן אבסטרקטי את הפונקציונאליות המוגדרת על ידה (הכוונה במחלקה פונקציונאלית היא למחלקה מגדירה מתודות ולא משתמש רק כמבנה המקבץ אוסף מאפיינים). לאחר החצנת פונקציונאליות זו, ניתן להחליף אצל הקליינטים את כל הקריאות המבוצעות למחלקות הקונקרטיות בקריאות למתודות המנשקים, כך שמבחינת הקליינטים האינטראקציה תהיה מול ה-Interface בלבד ולא מול מחלקה קונקרטית. טכניקת תכנות זו מאפשר להחליף בעתיד מימוש קיים של ה-Interface במימוש אחר באופן שכמעט ושקוף לקליינטים, שממילא עובדים מול ה-Interface ולא מול איזושהי מחלקה קונקרטית, כך שכל עוד המחלקה החדשה ממלאת את דרישות ה-Interface לקליינט זה כמעט ושקוף. למה כמעט שקוף ולא שקוף לחלוטין? כאן נכנס דפוס ה-**Factory** לתמונה.

אפילו אם כל הקריאות למתודות מצד הקליינט מבוצעות מול Interface ולא מול מחלקות קונקרטיות, בכדי לקבל איזשהו מופע של מחלקה המממשת את ה-Interface, כלומר, באיזשהו שלב חייבת להיות בקוד איזושהי קריאה לבנאי של מחלקה קונקרטית המממשת את המנשק. אם קריאה כזאת נמצאת בצד הקליינטים, אז עדיין יש תלות במחלקה קונקרטית: בכל פעם שיידרש להחליף את מימוש, יש למצוא את כל הקריאות לבנאים של אותה מחלקה ולהחליף אותן בקריאות לבנאים של המחלקה החדשה, בתקווה שיש תאימות כללית בפרמטרים של הבנאים מהמחלקות השונות. אמנם עדיין שיטת עבודה זו עדיין חוסכת את הצורך להחליף גם את הקריאות למתודות אולם היינו רוצים שחרור טוטאלי מהתלות של קליינטים במחלקות קונקרטיות. לשם כך אנחנו נעזרים בדפוס ה-**Factory**: אנחנו מחצינים את האחריות ליצירת מופעים של מחלקות קונקרטיות לאיזשהו גורם צד ג' (ה-Factory): הקליינטים יפנו אליו בבקשה לקבלת מופע של איזשהו Interface, והוא יהיה אחראי להחזיר מופע של איזושהי מחלקה קונקרטית המממשת Interface זה, באופן ששקוף לקליינטים. באופן זה, כל הקריאות לבנאים מרוכזות במקום אחד, כך שאם בעתיד יהיה צורך בהחלפת המחלקה המממשת, כל השינוי יהיה מבודד למקום אחד בלבד – ה-Factory, ומהצד של הקליינטים הדבר יהיה שקוף לחלוטין.

המערכת אינה מממשת את דפוסי ה-Factory הקלאסיים "by the book" כפי שמתוארים בספר של GoF – Gang Of Four, אלא בטכניקה שיותר דומה ל-"Simple Factory" המוצגת בספר DP של Freeman בהוצאת Head First. הסיבה למימוש זה היא שבמסגרת המערכת לא עלה הצורך לנהל משפחה של אובייקטים מאותו מפעל, ולכן לא היה צורך ב-Abstract Factory, וכן לא היה עניין בירושה ממחלקת מפעל כזו או אחרת ושינוי המימוש שלה להחזרת מופעים אלטרנטיביים, לכן גם לא היה צורך ב-Factory Method.

לפיכך המימושים שהוגדרו הם בעיקר בתצורה של מחלקת מפעל סטטית המגדירה מתודות סטטיות (סוג של Factory Methods) היוצרות מופע של איזשהו Interface ייעודי. בחלק מהמפעלים המופע המוחזר תלוי בבקשת הקליינט, למשל ב-ComposerFactoryהמקבל כארגומנט אסטרטגיית הלחנה מבוקשת ובהתאם יוצר מופע של Composer קונקרטי מתאים, ואילו בחלק מהמפעלים מוחזר פשוט מופע ישיר של המחלקה היחידה המממשת את ה-Interface: במקרים אלו כל עניין ה-Factory הוא לבצע deference של הקריאה לבנאי לטובת שבירת התלות של הקליינטים במחלקה הקונקרטית.

בפרויקט ה-Web לעומת זאת הוטמע שימוש בספריית Autofac המנהלת תלויות של קליינטים במנשקים ב-Inversion of Control (IoC) Container, ודואגת להזריק תלויות אלו בטכניקת Dependency Injection בזמן ריצה ע"י יצירת מופעים ב-Factory פנימי של הספרייה על-סמך קונפיגורציות מוגדרות מראש של יחסי התלות בין מנשקים והמחלקות המממשות אותם.

להלן רכיבי ה-Factory שהוטמעו במערכת –

1. [**MusicTheoryFactory**](#_מחלקת_מפעל_לישויות) **- מחלקה** המגדירה מתודות יצירה עבור מנשקי הישויות המוסיקליות השונות – משך-שהייה (IDuration), תו (INote), אקורד (IChord) ותיבה (IBar).
2. [**MidiFactory**](#_מחלקת_מפעל_לישויות_1) **- מחלקה** המגדירה מתודות יצירה עבור מופעים של המנשק IMidiFile המייצג קובץ MIDI.
3. [**ComposerFactory**](#_מפעל_ליצירת_מלחינים) **- מחלקה** המגדירה מתודת יצירה עבור מופעים קונקרטיים של המחלקה האבסטרקטית Composer בהתאם לאסטרטגיית הלחנה מבוקשת המוגדרת כערך ב-Enum של CompositionStrategy ומועברת למתודה המפעל ארגומנט.
4. **Autofac Dependency Injection - בפרויקט ה**-Web בקובץ ההגדרות הגלובאליות Global.asax.cs, במסגרת המתודה Application\_Start במחלקה MvcApplication הוטמעו רישומי תלויות ב-IoC המנוהל ע"י ספריית Autofac כדלקמן –

// 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, הן שבזמן ריצה, תחת הקונטקסט המוגדר, עבור כל תלות של קליינטים במנשק IUnitOfWork, ספריית Autofac תדאג לספק מופע של המחלקה הקונקרטית EFUnitOfWork, ובאותו אופן, עבור כל תלות במנשק ISongRepostiory, ספריית Autofac תדאג לספק מופע של המחלקה הקונקרטית SongRepository. לפיכך, תוך שימוש בספרייה זו בפרויקט ה-Web, הבקרים (Controllers) לא צריכים להכיר את המחלקות הקונקרטיות בשכבת הגישה לנתונים (DAL). כל שהם צריכים זה לעבוד מול ה-Interfaces של מנהל הישויות (האינטגרטור IUnitOfWork שאחראי על האינטראקציה מול ה-DB) ושל מאגרי הישויות שנדרש לאחזר ו/או לעדכן (IRepository לצורך העניין). ספריית Autofac מגדירה מימוש משלה לדפוס ה-Factory ומחזירה מופעים מתאימים באופן ששקוף לקליינט של יישום ה-Web, כך שבהקשר הזה אין לו מושג האם מדובר ב-EntityFramework או חבילת ORM אחרת או אולי אפילו שימוש ישיר ב-API של ADO.NET.

למעשה, ניתן היה (ואולי אף מוטב היה) לעשות שימוש רוחבי בספריית Autofac בכל הפרויקטים במערכת ולחסוך את הצורך בהגדרת מפורשת ידנית של מחלקות ה-Factory, שכן ספרייה זו מספקת פונקציונאליות נוספת כגון קביעת ה-Scope של המופעים (למשל מופע יחיד עבור כלל הבקשות – Singleton, או מופע חדש לכל בקשה – Prototype), ריכוז כלל התלויות במקום אחד מרכזי במקום במחלקות Factory שונות, ואולי הכי חשוב: מבצעת Dependency Injection אצל הקליינטים וחוסכת מלכתחילה את הצורך בפנייה מפורשת ל-Factories.

עם זאת, הפרויקטים השונים במערכת פותחו בזמנים שונים: חלק מה-Factories במחלקת הלוגיקה העסקית הוגדרו טרם הגדרות פרויקט ה-Web בשכבת התצוגה, כך שהטמעת ה-Autofac בדיעבד מצריכה עבודה נוספת, מה שגם שהקונפיגורציה בפרויקט Class Library של שכבת הלוגיקה העסקית שונה מזו שמוגדרת מפרויקט Web של ASP.NET MVC. נקודה נוספת לטובת התצורה הקיימת בשכבת הלוגיקה העסקית היא שההגדרות דרך מחלקות ה-Factory המפורשות הרבה יותר ברורות, להבדיל מאוסף הגדרת Dependencies באיזה קובץ קונפיגורציה שמעלים את כל הלוגיקה ונראה כמו "קסם" מבלי שברור בדיוק איך הדברים עובדים מאחורי הקלעים, ומתי ואיפה נוצרים המופעים ומה ה-scope שלהם.

## דפוסים מבניים (Structural Patterns)

### דפוס מעטפת מתאמת (Adapter Pattern)

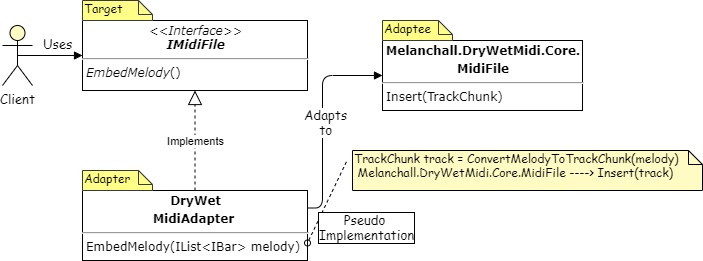
דפוס ה-Adapter יושם במערכת [בתת-שכבת שירותי ה-MIDI](#_תת-שכבת_שירותי_מידי): פלטפורמת ה-.NET אינה מספקת ספריות סטנדרטיות ב-C# לתמיכה ואינטראקציה עם פרוטוקול MIDI. המערכת צריכה שירותים בסיסיים כגון – קריאה וכתיבה של קובץ MIDI, נגינת קובץ והחלפת רצועות בקובץ. לטובת צריכת שירותים אלו נעשה שימוש בספרייה חיצונית בשם DryWetMIDI –

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

ספרייה זו עוטפת את הפרטים הטכניים (low-level) של הפרוטוקול ומספקת מנשק פשוט יחסית לעבודה מול קובצי ה-MIDI ב-high-level, אולם גם מנשק זה דורש לא מעט עיבוד והתאמות כדי שניתן יהיה לעבוד מולו במערכת. למשל: בשביל להטמיע מנגינה חדשה שהולחנה כרצועה בקובץ MIDI, יש להמיר את כל התווים כפי שמיוצגים במערכת (ע"פ התיאור בסעיף [ייצוג ישויות מוסיקליות](#_ייצוג_ישויות_מוסיקליות)) לאירועי MIDI כפי שמיוצגים ע"י הספרייה החיצונית. זה כולל המרות בזמנים של משכי שהייה, בטיפוסים, שמות ועוד. ספריות אחרות העובדות מול פרוטוקול MIDI עשויות לספק מנשק אחר לגמרי לקליינטים. לפיכך, היה חשוב למנוע צימוד (coupling) בין קוד הליבה של המערכת לבין הקוד של הספרייה החיצונית, ולעבוד מול אבסטרקציה (IMidiFile interface לצורך העניין), כך שאם בעתיד יעלה הצורך להחליף את הספרייה הקיימת באחרת, הדבר לא צריך שינוי אצל הקליינטים כל עוד הספרייה החדשה תדאג לעמוד בתנאי האבסטרקציה שהוגדרו.

דפוס ה-Adapter מתאים מאוד עבור תרחיש זה: ה-IMidiFile (המנשק של המערכת שמייצג קובץ MIDI) הוא על תקן ה-Target Interface, המחלקה MidiFile מהספרייה החיצונית היא על תקן ה-Adaptee, והמחלקה DryWetMidiAdapter היא על תקן ה-Adapter. הקליינטים הם תת-השכבות/שכבות אחרות העושים שימוש במנשק הנח IMidiFile, דוגמת CompositionContext בשכבת הלוגיקה העסקית, ובקר ה-CompositionController בשכבת התצוגה בפרויקט ה-ASP.NET MVC.

להלן תרשים מחלקה הממחיש שימוש טיפוסי ב-Adapter: קליינט מבקש ממנשק היעד IMidiFile להטמיע מנגינה בקובץ MIDI. מחלקת ה-Adapter (DryWetMidiAdapter) ראשית ממירה את המנגינה מהייצוג הפנימי לייצוג החיצוני של ספריית ה-Adaptee, הלוא היא Melanchall.DryWetMidi. לאחר ההמרה, מחלקת ה-Adapter יכולה לעשות שימוש ישיר במנשק שמחלקת ה-Adaptee מהספרייה החיצונית מספקת, עבודה עם Collection של רצועות ופשוט להוסיף את הרצועה החדשה (למען פשטות הושמטו פרטים אחדים כגון פרמטר אינדקס) –



הערות: זוהי כמובן גרסת המימוש של object adapter (ולא class adapter), שכן שפת C# (להבדיל מ-C++ למשל), אינה תומכת בירושה מרובה. כמו כן, מעבר לאדפטציה למחלקה הספציפית MidiFile של הספרייה החיצונית, ה-Adapter עוטף שירותים נוספים של הספרייה הנתונים ב-low-level יחסי, ומנגיש אותם ב-high-level, לכן מתפקד גם קצת כ-Facade לקליינטים.

## דפוסים התנהגותיים (Behavioral Patterns)

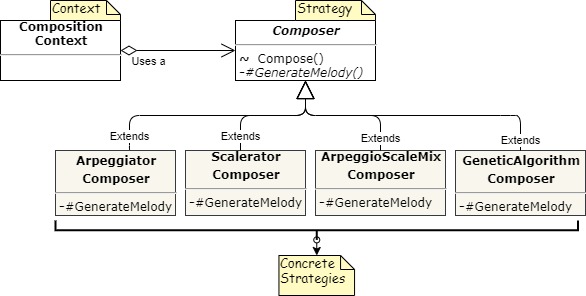
### דפוס אסטרטגיה אלגוריתמית (Strategy Pattern)

האלגוריתם המרכזי שהמערכת משתמשת בו להלחנת מנגינות הוא האלגוריתם הגנטי המתואר בסעיף [הלחנה עם אלגוריתם גנטי](#_הלחנה_עם_אלגוריתם). עם זאת, אין זה האלגוריתם היחיד המשתמש באוטומציה וכוח עיבוד לטובת הלחנה. בתחום ב-[Algorithmic Composition](https://en.wikipedia.org/wiki/Algorithmic_composition) קיימים מגוון אלגוריתמים ושיטות נוספות לגמרי, כגון אלגוריתם המבוססים על חוקי-דקדוק ושפה, אלגוריתמים מבוססים מערכות-מומחה, ואלגוריתמים המבוססים על מודלים מתמטיים כגון שימוש בתהליכים סטוכטיים אי-דטרמיניסטיים.

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

בכדי לאפשר החלפה דינאמית של אלגוריתם ההלחנה, הוטמע דפוס ה-Strategy: מחלקת האב-האבסטרקטית Composer מגדירה מתודה אחת חיצונית Compose שאותה הקליינטים מכירים, ומתודה נוספת – GenerateMelody, פרטית ואסטרקטית, הנקראת מתוך Compose, שאחראית לממש את ההלחנה בפועל. מחלקות קונקרטיות מממשות מתודה זו עם אלגוריתם ההלחנה המבוקש, ומבחינת הקונטקסט שמבקש לבצע את ההלחנה, זה שקוף לו, הוא תמיד קורה לאותה מתודת Compose מאיזשהו מופע של Composer ללא מודעות מהי המחלקה הקונקרטית שמבצעת את ההלחנה בפועל.

להלן תרשים מחלקה המתאר את רכיבי המערכת המשתתפים בהטמעת דפוס ה- Strategy –



מטעמי פשטות, הדיאגרמה מסתירה פרטי מימוש טכניים מסוימים: לטובת החלפה דינאמית בזמן ריצה של אלגוריתם ההלחנה המבוקש, המערכת מגדירה אוסף קבועים המייצגים את אלגוריתמי ההלחנה השונים המוגדרים במערכת ב-Enum בשם [CompositionStrategy](#_CompositionStrategy_Enumeration). קונטקסט ההלחנה מגדיר במתודת ההלחנה פרמטר אופציונאלי שבו קליינטים יכולים לציין את אסטרטגיית ההלחנה המבוקשת ע"י ציון ערך ה-Enum המתאים. (אם לא מצוין אף ערך, המערכת בוחרת כברירת מחדל את האלגוריתם הגנטי). קונטקסט ההלחנה פונה עם אסטרטגיית ההלחנה המבוקשת ופונה למחלקת [מפעל ליצירת מלחינים](#_מפעל_ליצירת_מלחינים) (ComposerFactory) ומבקש ממנה להחזיר לו מופע של Composer בהתאם לאסטרטגיה המבוקשת. המפעל מחזיר מופע של מחלקה קונקרטי המממשת את Composer, והקונטקסט משתמש בה להלחנה באמצעות מתודת Compose. דפוס עיצוב זה הופך את הקוד לגנרי לחלוטין ביחס לאלגוריתם ההלחנה: בשום שלב לא קליינטים ולא לקונטקסט ההלחנה יש שמץ של מושג מהי המחלקה הקונקרטית המוציאה לפועל את מלאכת ההלחנה. כל העבודה של הקליינטים נעשית מול הקונטקסט, וכל העבודה של הקונטקסט מבוצעת מול המנשק המוגדר ע"י המחלקה האבסטרקטית Composer. כך שאם בעתיד תוגדר מחלקה נוספת היורשת מ-Composer, ויוגדר עבורה ערך מתאים ב-Enum האסטרטגיות, קליינטים יוכלו לציין ערך Enum זה, ולקונטקסט זה יהיה שקוף לחלוטין!

### דפוס תבנית אלגוריתמית (TemplateMethod Pattern)

כאמור, אלגוריתם ההלחנה המרכזי במערכת הוא האלגוריתם הגנטי. למעשה אין זה מדויק לומר **ה**אלגוריתם הגנטי עם 'הא הידיעה': אלגוריתם גנטי הוא שם כולל ל**משפחה** של אלגוריתמים, אשר המשותף להם הוא שימוש בשיטות חיפוש היוריסטיות תוך התבססות עמוקה על עקרונות ותובנות מתהליכים אבולוציוניים המתרחשים בטבע. אלגוריתמים גנטיים שונים עשויים להכיל מימושים שונים לחלוטין, פירוט נוסף על כך ניתן למצוא 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 מוסכמה לגבי איזושהי תבנית כללית של אלגוריתם גנטי טיפוסי. להלן שלד האלגוריתם –

1. אתחל דור ראשון
2. בצע –
   1. שחלוף
   2. מוטציה
   3. שיערוך
   4. בחירת דור המשך
3. בדוק האם מתקיים תנאי העצירה –
   1. אם כן, החזר כפלט את המועמד(ים) הכשיר(ים) ביותר וסיים;
   2. אחרת, חזור לשלב 2.

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

* + **אתחול**: האם לבצע אתחול מכוון על-סמך ידע מוקדם, או אתחול אקראי לחלוטין ?
  + **שחלוף**: איזו שיטת שחלוף לבצע? כיצד לבחור את המועמדים מהאוכלוסייה שישתתפו בשלב השחלוף? כיצד לקבוע את מספר נקודות השחלוף והמיקום שלהן?
  + **מוטציה**: איזה מוטציות לבצע? האם להגדיר מוטציות ניטרליות לגמרי, או האם המוטציות צריכות להיות מודעות למצב והתוכן של המועמדים? כיצד לקבוע את המועמדים שיעברו מוטציה? האם ובאיזה תנאים עדיף לבצע שינויים נרחבים על-פני שינויים מזעריים?
  + **שיערוך**: מהי הדרך הטובה ביותר לאמוד את טיב-איכות הפתרונות המועמדים? כיצד יש להתמודד עם tradeoffs בין מדדי הצלחה מנוגדים?
  + **בחירת דוך המשך**: איזה חלק יחסי מהאוכלוסייה יש לסנן/לבחור? האם לבחור תמיד את המועמדים הטובים ביותר, או לתת הזדמנות גם למועמדים אחרים בעלי פוטנציאל? האם לבחור תמיד רק מועמדים מהדור האחרון או גם מועמדים מדורות קודמים?
  + **תנאי עצירה**: כיצד לקבוע מתי לעצור? האם ניתן להעריך איזשהו מדד על מגמת התכנסות של האלגוריתם? האם לעצור אחרי מספר קבוע של איטרציות? האם יש איזשהו סף איכות שאם מגיעים אליו ניתן לעצור ?

החלטות שונות בסוגיות אלו מובילות להתפצלויות שונות לגמרי ומגדירים מימושים שונים שיכולים להביא לידי פלטים שונים לגמרי. בכדי לספק תמיכה בשלד האלגוריתם הטיפוסי ויחד עם זאת לאפשר גמישות במימוש השלבים השונים של האלגוריתם, המחלקה GeneticAlgorithmComposer מגדירה את מגוון מתודות הניהול של השלבים השונים כ-virtual בכדי לאפשר למחלקות לרשת ממנה ולממש אותן באופן אלטרנטיבי. מימוש זה אינו המימוש הקלאסי (("by the book" של דפוס ה-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 ירושה ממחלקה זו ושינוי הקומבינציה של המתודות הפנימיות שמתודות הניהול מפעילות – למשל להחליף במתודה SelectNextGeneration את הקריאה ל-PlusSelection בקריאה ל-RouletteWheelSelection.

## תבניות ודפוסים ארכיטקטוניים

מעבר לדפוסי העיצוב המתוארים בספר של GoF, מוטמעים במערכת דפוסים נוספים המתוארים בספרו של Martin Fowler בשם Patterns of Enterprise Application Architecture, המתאר דפוסים המסייעים בעיצוב ארכיטקטורה חכמה של יישומים ומערכות גדולות רחבות היקף:

<https://bit.ly/34Einlr>

### מודל-תצוגה-בקרה ((MVC

שכבת התצוגה של המערכת הוגדרה בפרויקט Web בפלטפורמת ה-ASP.NET MVC. שם הפלטפורמה מרמזת על שם הדפוס שעליו היא מושתתת: **MVC** – **M**odel-**V**iew-**C**ontroller. הדפוס כבר מוגדר במסגרת הפלטפורמה, וכל שנוצר הוא להיצמד אליו –

* + ה-**Models** הרלוונטיים ל-Persistence הועברו מפרויקט ה-Web לפרויקט ה-DAL, כך שניתן יהיה לעשות בהם שימוש חוזר גם בפרויקטים אחרים (למשל אם יהיה צורך בכך ניתן יהיה לעשות שימוש במודלים אלו בפרויקט WPF המפותח בתבנית העיצובית של MVVM. מעבר למודלים אלו, הוגדרו בפרויקט ה-Web מודלים ייעודיים לריכוז נתונים רלוונטיים לתצוגה, כגון תיאורים ידידותיים למשתמש של קבועים, רשימות ערכים לבחירה ונתוני הרשאה. מודלים אלו רוכזו בתיקיה ייעודית ViewModels. המודלים מכילים רק לוגיקה סביב אחזור הנתונים למודל ללא לוגיקה עסקית. המודלים אינם מודעים לא לבקרים ולא ל-Views.
  + ה-**Views** מגדירים תבניות המורכבות מ-HTML, קוד C#, וסקריפטים ב-javascript, ומנוע של .NET אחראי לפרש תבניות אלו וליצור מהם דפי HTML לתצוגה לאחר שהוא משבץ בהם נתונים מהמודלים המועברים ל-Views ע"י הבקרים. ה-Views מגדירים איזה Model הם מצפים לקבל, אך אינם מודעים לבקר ואינם מכילים לוגיקה למעט לוגיקה הרלוונטית לאופן התצוגה (למשל ההחלטה אם להציג הודעות מערכת מהשרת בצבע אדום).
  + ה-**Controllers** ומשמשים כנתבים או אינטגרטורים, המקבלים בקשות ממשתמשי הקצה, אוספים נתונים רלוונטיים למודלים, מעבירים את המודלים ל-View המתאים ומחזירים למשתמשי הקצה את ה-View המעובד עם הנתונים המשובצים מהמודל (או לחלופין מחזירים ערך אחר במקום View דוגמת מסמך PDF / קובץ MIDI בהתאם לאותה הבקשה).

פירוט נוסף על מימוש הפרויקט ניתן למצוא בסעיף [שכבת התצוגה](#_שכבת_התצוגה_(PL)). אינטראקציה לדוגמה בין ה-Model, ה-View וה-Controller ניתן לראות בסעיף [אינטראקציה בין רכיבי ה-MVC](#_אינטראקציה_בין_רכיבי).

### מיפוי אובייקטי-רלציוני (ORM)

שכבת הגישה לנתונים ממומשת מאחורי הקלעים באמצעות ספריית EntityFramework. ספרייה זו מממשת את תבנית ה-Object-Relational-Mapping (ORM), האחראית על גישור הפערים /בין ייצוג הישויות העסקיות במודל הקונספטואלי של עולם האובייקטים בפרדיגמת OOP, לבין ייצוגם במודל הפיזי בעולם מסדי-הנתונים הרלציוניים. למשל – בעולם ה-OOP יש יחסי ירושה, ואילו בטבלאות ב-DB אין. קיימים גם פערים בטיפוסי שדות (string לעומת varchar או nvarchar, ייצוג משתנים בוליאניים, אפשור/אי-אפשור ערכי null לעמודות נומריות וכד'). פער נוסף הוא צורך בטבלת קשר ב-DB עבור מידת קשר של "Many-to-many"", כאשר בעולם האובייקטים ניתן פשוט להגדיר במחלקה אוסף ישויות 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 (הקונספטואלי מול הפיזי), ומספקים מנשק נח המאפשר לקליינטים לנהל אינטראקציה מול ה-DB הפיסי בשפת OOP טהורה בטרמינולוגיה ובסמנטיקה של הישויות בעולם האובייקטים עם מחלקות ומתודות וללא שימוש ב-SQL. שימוש זה חוסך לא מעט עבודה, ומפשט את הקוד משמעותית.

פירוט נוסף על אופן הטמעת כלי ה-ORM של EntityFramework במערכת ניתן למצוא בפרק [מיפוי בין אובייקטים לרשומות (EntityFramework ORM)](#_מיפוי_בין_אובייקטים)

### הפשטת אינטראקציה מול DB (Repository & Unit Of Work)

דפוס ה-**Repository** משמש כמאגר זמן ריצה (מנוהל זיכרון ה-RAM) של ישויות, לרוב מאותו טיפוס, המספק מעין מנשק מילוני (כגון CRUD – Create, Read, Update, Delete) לאחזור ועדכון ישויות המנוהלות במאגר זה, תוך הסתרת האינטראקציה מול מדיום האחסון הפיסי שבו הישיות מאוחסנות בפועל.

דפוס ה-**Unit Of Work** משמש לניהול עדכונים מצטברים המבוצעים באיזשהו Session בזמן ריצה אל מול ה-Database שעדיין לא נרשמו פיסית: רישום פיסי (ביצוע Commit) על כל שינוי בנפרד יגרור מחיר כבד בביצועים, ולכן עדיף לנהל 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 ההכנסה של INSERT טרם פקודת העדכון UPDATE). זהו תפקידו של ה-Unit of Work, לנהל מעקב אחר כל הישויות הרלוונטיות ל-Persistence, לרשום מי עודכן ומתי, לאסוף את כל השינויים האלו המתבצעים באיזושהי יחידת זמן /Session / טרנסאקציה ולבסוף לפנות אל מדיום האחסון הפיסי ולבצע את הרישום (Commit) בפועל.

בגדול, ספריית ה-Entity Framework המממשת את דפוס ה-ORM שתואר בסעיף קודם, מממשת איזושהי קומבינציה של דפוסי ה-Repository וה-Unit of Work באמצעות המחלקה DbContext ומאפייני ה-DbSet<TEntity> שלה. להלן תקציר המחלקה DbContext מתוך התיעוד של Microsoft (<https://bit.ly/34GfD75>) –

*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 "בגדול" (לפחות איזושהי קומבינציה שלו). עם זאת, שימוש ישיר בישויות ה-DbContext וה-DbSet יוצר תלות ישירה של הקליינטים (במקרה הזה הקליינט הוא שכבת הגישה לנתונים) בספריית ה-Entity Framework. אם בעתיד נרצה להחליף ספרייה זו בכלי ORM אחר, דוגמה NHibernate או אפילו החלפה לעבודה ישירה מול ה-API של ADO.NET, הדבר יצריך פחות או יותר שכתוב מחדש של שכבת הגישה לנתונים מאחר שכל הקוד שלה יהיה מעורה ותלוי ב-API של ה-Entity Framework.

בכדי לשבור תלות זו, הוגדרה רמת הפשטה **נוספת**, מעל ה-Entity Framework, המגדירה מנשקים עבור שני הדפוסים שלעיל –IRepository (עבור דפוס ה-Repository) ו-IUnitOfWork (עבור דפוס ה- Unit Of Work), ושכבת הגישה לנתונים עובדת רק מול המנשקים האבסטרקטיים, ואינה ומודעת למימוש הקונקרטי מאחורי הקלעים.

בפועל, המימוש מבוצע כמובן באמצעות Entity Framework, שכאמור כבר מספקים תמיכה בדפוסים אלו, לכן כל מה שנדרש הוא להגדיר מחלקות חוצצות שפשוט לוקחות את הבקשות של הקליינטים ומעבירה אותן הלאה ב-delegation למחלקות הרלוונטיות ב-Entity Framework היודעות לעשות את העבודה: DbContext מבצע פעולות של IUnitOfWork כגון שמירת השינויים ב-DB (Commit), וה--DbSetס שלו מבצעים פעולות של ה-Repository כמו אחזור רשימת ישויות מה-DB, חיפוש רשומה לפי מפתח (או פרדיקט אחר כלשהו) ומחיקת ישויות 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)](#_שכבת_הגישה_לנתונים), בתתי-פרקים של [מאגרים (Repositories)](#_מאגרים_(Repositories)) ו-[ניהול יחידות עבודה (Unit Of Work)](#_ניהול_יחידות_עבודה).

# תיאור שינויים עתידיים

סעיף זה מכיל תיאור של מספר שינויים עתידיים אפשריים במערכת והסבר על הטמעתן. נחלק את השינויים לשתי קטגוריות – האחת: שינויים תשתיתיים – שקשורים לפלטפורמה וסביבות ומערכות חיצוניות שהמערכת תלויה בהן כדי לרוץ, אך אינם קשורים לפונקציונאליות או הלוגיקה העסקית של המערכת, והשנייה: שינויים אפליקטיביים/פונקציונאליים – שינויים במערכת עצמה, דהיינו בלוגיקה העסקית המוגדרת בכדי לספק פונקציונאליות שונה.

## שינויים תשתיתיים

### שינויים תשתיתיים הקשורים ל-DB

#### החלפת מסד-נתונים

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

אם השרת שהם מספקים גם הוא של SQL Server מבית Microsoft, כל שיידרש הוא הסבת תוכן ה-DB הקיים אל ה-DB החדש (העתקת הסכמה ותוכן כל הטבלאות), ועדכון פרטי החיבור ב-Connection String בקובצי הקונפיגורציה (קובצי ה-XML של ה-App.config, Web.config וכד') בהתאם לשרת ה-DB החדש.

אם שרת ה-DB הוא שרת אחר, למשל שרת MySQL מבית Oracle או שרת PostgreSQL, כל שיידרש בנוסף לצעדים שלעיל, בהנחה שקיימת מחלקת Provider Data מתאימה ב-.NET (אנלוגי ל-JDBC Driver ב-Java), הוא גם ייבוא ה-Assemblies של ספק נתונים זה, ציון ה-namespaces שלו ועדכון הפרטים שלו בקובצי הקונפיגורציה ב-XML-ים שלעיל במקום פרטי ספק הנתונים של SQL Server.

#### החלפת טכנולוגיית ORM

המערכת משתמשת בספריית ה-Entity Framework בגרסה 6 שלה, בתור טכנולוגיית ה-ORM המבצעת את המיפוי, המרות וגישור בין הרשומות השטוחות בטבלאות ב-database שבמודל הפיסי לבין האובייקטים המייצגים אותם במודל הקונספטואלי. גרסה 6 נחשבת כיום גרסה ישנה: פלטפורמת ה-.NET Framework נמצאת היום במגמת שינוי ומיגרציה לטכנולוגיות Cross-Application שאינן תלויות בפלטפורמה או מערכת הפעלה (להבדיל למשל מטכנולוגיות ישנות שפותחו על-ידן כגון WPF שמוגבלת אך ורק ליישומי Windows). טכנולוגיות אלו מפותחות תחת פלטפורמת ה-.NET Core, וגם גרסת ה-Entity Framework העדכנית ביותר, שבעבר טרם שחרורה יוחס לה גרסה מספר 7, נכתבה מחדש כדי להתאימה ל-.NET Core, ויצאה תחת השם Entity Framework Core.

טבעי יהיה לשדרג לגרסה העדכנית ביותר, או לכל הפחות, להחליף את הטכנולוגיה הנוכחית הנחשבת כישנה, באחת אחרת, דוגמת NHibernate. מאחר שליבת שכבת הגישה לנתונים מוגדרת לעבודה אך ורק מול מנשקים אבסטרקטיים (IUnitOfWork ומופעי IRepository), אין תלות בטכנולוגית Entity Framework, כך שבכדי להחליף את טכנולוגיית ה-ORM הקיימת, כל שיידרש הוא כתיבת מחלקות מתאימות למימוש המנשקים האבסטרקטיים שלעיל תוך delegation ואדפטציה מתאימה למחלקות הטכנולוגיה החדשה, ועדכון הקונפיגורציה של הזרקת התלויות (Dependency Injection) של ספריית Autofac בקובץ Global.asax.cs כך שה-Factory של הספרייה יחזיר מופעים של המחלקות החדשות במקום אלו הקיימות.

### שינויים בפלטפורמה

שכבת התצוגה הנוכחית של המערכת ממומשת בפרויקט Web בפלטפורמת ASP.NET MVC כאתר אינטרנט. אחד היתרונות של יישומי Web שהוא נגיש מכל פלטפורמה (בהתאם יש לכתוב את הקוד בצד ה-client כך שיהיה מותאם לממדי המכשירים השונים, למשל תוך שימוש נכון בספריות Bootstrap).

אולם, נניח שיהיה צורך גם בפיתוח אפליקציה native לאיזשהו מכשיר, למשל כאפליקציה לאנדרואיד ו/או iOS. ובכן, לאור החלוקה של המערכת לשכבות, עיקר המאמץ יהיה פיתוח מנשק המשתמש (GUI) ל-smartphone שיחליף את שכבת התצוגה הקיימת. כל הלוגיקה העסקית המתוחכמת של האלגוריתם הגנטי, ייצוגי תווים ושימוש ב-DB, כל אלו יוותרו ללא שינוי. הם רק שירותים ששכבת התצוגה יכולה לצרוך ואינם תלויים בקליינט שצורך את השירותים הללו. נניח למשל שנפתחים אפליקציית בטכנולוגיות Xamarin, אזי שכבת תצוגה שמגדירים שם יכולה לצרוך את שירותי ההלחנה של המערכת תוך שימוש במחלקה CompositionContext, ואת שירותי ה-Database לשמירה ואחזור של שירים קיימים תוך שימוש בשכבת הגישה לנתונים.

אם יש צורך בצריכת השירותים ע"י אפליקציה הכתובה בטכנולוגיה אחרת לגמרי, למשל אפליקציה הכתובה ב-python המיועדת לרוץ על מערכות הפעלה של Linux, עדיין ניתן יהיה בקלות להסב את שירותי ההלחנה ע"י עטיפתם ב-API (למשל בטכנולוגיית ASP.NET Web API 2 או ASP.NET .Core), ואז בנוסף לאספקת שירותים אלו למשתמשי קצה באמצעות דפי HTML באתר אינטרנט, המערכת תספק גם end-points: נתיבי URL שניתן לפנות אליהם ולקבל את השירות ב-API, שאינו תלוי פלטפורמה.

## שינויים אפליקטיביים/פונקציונאליים

### הוספת אלגוריתם הלחנה

המערכת מגדירה כעת אלגוריתם מרכזי יחיד – האלגוריתם הגנטי. אמנם מוגדרים מספר מחלקות נוספות במקביל למחלקת האלגוריתם הגנטי, אולם הן די מנוונות: האלגוריתם שלהם פשוט מחולל רצפי צלילים בסיסיים על מתוך צלילי האקורדים והסולמות המתאימים למהלך ההרמוני של השיר ללא יישום לוגיקה נוספת.

נניח שנרצה לשלב במערכת אלגוריתם נוסף שעושה עבודה יסודית בדומה לאלגוריתם הגנטי, למשל איזה אלגוריתם Machine Learning שמנתח מאגר יצירות לדוגמה הניתנות לו כקלט ונחשבות כ-"טובות" ומלחין מנגינה בהשראה ובסגנון של יצירות אלו. כבודה של מורכבות האלגוריתם במקומה, אבל בהינתן אלגוריתם שכזה, כלומר בהנחה שכבר קיים מימוש של האלגוריתם, העובד על ישויות מוסיקליות ע"פ הייצוג 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 המתודה האבסטרקטית GenerateMelody.

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

שינוי זה מתאפשר הודות למימוש דפוס העיצוב Strategy.

### עדכון האלגוריתם הגנטי הקיים

נניח שנרצה להכניס שינויים באלגוריתם הגנטי הקיים. נציע שינוי רציונלי שטבעי שיהיה בו צורך בעתיד.

#### קיבוע צלילים בנקודות עוגן

בתור נגן גיטרה חשמלית חובב, לא פעם באלתור או חיבור מנגינה חדשה יש לי רעיון כללי על איזשהן נקודות עוגן במנגינה: הצליל הראשון שבו אני מעוניין לפתוח את המנגינה (או סדרת הצלילים) , הצליל האחרון שאנגן בכדי לסיים את המנגינה, ועוד מספר נקודות שיא מקומיות. אח"כ אני מנסה לחבר את הקשרים שבין הנקודות, כלומר למצוא איך להגיע מהצליל המתנגן בנקודת הפתיחה לצליל בתחנת הביניים הבאה באופן המשקף הכי טוב את מה שאני מחפש לשמוע ו/או לשדר. לפיכך הייתי רוצה שהמערכת תאפשר לי בתור משתמש לקבע נקודות עוגן מבחינת גובה צליל ו/או משך שהייה, שמובטח לי שיישארו כמות שהן במנגינה החדשה שתיווצר, כלומר האלגוריתם רשאי להחליף את כל הצלילים שבין הנקודות, אך באותן נקודות מוגדרות התווים המקוריים יישמרו.

כיצד נוכל לממש Feature שכזה במסגרת האלגוריתם הגנטי הקיים?

הודות לתצורה הנוכחית של האלגוריתם, המגדירה שלד כללי המפעיל מתודות ניהול המוגדרות כ-virtual, וחלוקה למודולים נפרדים האחראים לממש מתודות אלו, ניתן בקלות יחסית לשנות ו/או להרחיב את הלוגיקה הקיימת של האלגוריתם תוך שימוש בדפוס Template Method: נגדיר מחלקה חדשה היורשת מהמחלקה GeneticAlgorithmComposer, והיא תוכל לבצע override למתודות הרלוונטיות של האלגוריתם הגנטי, לרבות מתודת האתחול הראשית InitializeCompositionParams המוגדרת במחלקת-האב האבסטרקטית Composer כמתודת virtual, כך שגם אותה ניתן לדרוס במחלקות יורשות ע"פ הצורך.

לאחר הגדרת מחלקה היורשת מ- GeneticAlgorithmComposerהשאלה היא איפה המקום הכי טוב "להתערב".

ראשית, בכדי לציין את מיקומי נקודות העוגן, ניתן להיעזר בפרמטר הכללי params object[] customParams המועבר למחלקה במסגרת המתודה InitializeCompositionParams. מאחר שפרמטר זה מקבל מערך בכל אורך סופי שנבחר של משתנים מכל טיפוס שנבחר, נוכל להגדיר 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 השייך לתיבה המיוצגת ע"י צומת האב ברמה שמעל. תצורה זו של אופן ייצוג נקודות העוגן ומבנה הנתונים שישמש לכך – אלו הם פרטים טכניים שכבודם מונח במקומם, אך הם לא משפיעים על הארכיטקטורה והמשך שילוב השינוי המבוקש במערכת, לכן לא נידון בהם עוד, אלא רק נניח שהפרמטר customParams מכיל את נקודות העוגן, כך שהמחלקה החדשה היורשת מ- GeneticAlgorithmComposer יכולה לאחסן נקודות אלו כרצונה ויש לה גישה לנקודות אלו בכל שלב באלגוריתם.

שנית, אחרי שיש לנו את הקלט הרלוונטי, יש לבחון כיצד ואיפה להתייחס אליו במסגרת התצורה הקיימת. דרך אחת, היא לעבור במחלקה היורשת על כל המתודות שמעדכנות מנגינות – מתודות אתחול, מתודות מוטציה ומתודות שחלוף ולעדכן אותם באופן שיתייחס לנקודות העוגן וישמור עליהם מפני כל שינוי. דרך זו נשמעת ארוכה, מסורבלת ומועדת לפורענויות (למשל באגים הנובעים מפספוסים של מגוון קומבינציות ומקרי קצה). דרך אלטרנטיבית, עצלנית, קצרה ופשוטה יותר, היא כן לאפשר לאלגוריתם לשנות את כל הצלילים לרבות נקודות העוגן כפי שהיה עד כה, ובסיום כל מחזור (איטרציה) של האלגוריתם, להריץ מתודת מוטציה חדשה שתוגדר במחלקה החדשה העוברת על אוכלוסיית המנגינות החדשות ומטמיעה בהן את הערכים המבוקשים בנקודות העוגן. לחלופין, ניתן לשפר ביצועים ולבצע הטמעה זו רק בסיום כלל המחזורים בסוף האלגוריתם בטרם החזרת הפלט הסופי לקליינטים, אולם אז שלב השערוך יהיה פחות מדויק שכן הוא ייקח בחשבון את המנגינות שממילא עתידות להשתנות טיפה לאחר הטמעת נקודות העוגן, לכן עדיף לבצע זאת בכל מחזור בנפרד לאחר שלב המוטציה ולפני שלב השערוך. ניתן למשל לעשות זאת מתוך מתודת ניהול המוטציות Mutate , לאחר ריצת כל יתר המוטציות.

# נספח: רקע תאורטי של המוסיקה המערבית

סעיף זה סוקר בקצרה מונחים בסיסיים מהטרמינולוגיה המוסיקלית שמוזכרים 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 נקצר ונאמר פשוט 'צליל' כאשר הכוונה היא לגובה צליל. **מרווח** הוא היחס שבין שני (גבהי) צלילים. מקובל למדוד יחס זה ביחידות של טון(או חצאי-טון). **חצי-טון** הוא המרווח הקטן ביותר הנמצא בשימוש במוסיקה הטונאלית המערבית, והוא מבטא יחס של  בין תדירויות גלי הקול שני צלילים הסמוכים זה לזה (דהיינו צלילים הנמצאים במרחק של חצי-טון זה מזה). בתרבויות אחרות (למשל במוסיקה הודית ובמוסיקה ערבית) משתמשים אף במרווחים קטנים יותר של רבעי-טונים.

מרווח של שנים-עשר חצאי טונים נקרא **אוקטבה**. בהתאם ליחס בין תדירויות גלי קול המוגדר ע"י מרווח של חצי-טון, מרווח האוקטבה מגדיר יחס של , דהיינו תדירות כל צליל היא בדיוק פי-שתיים מתדירות הצליל הנמוך ממנו באוקטבה. צלילים הנמצאים במרווח של אוקטבה בדיוק זה מזה דומים מאוד אחד לשני, עד כדי כך שבמוסיקה המערבית מסווגים אותם לאותו צליל בסיסי. בהתאם לכך, מבחינים בין סה"כ שנים-עשר צלילים בסיסיים, כך שכאשר מתייחסים לאיזשהו צליל בסיסי, הכוונה היא בעצם למשפחה או מחלקה של צלילים, המכילה איזשהו צליל בסיסי מסוים ואת כל הצלילים הנבדלים ממנו בכפולות שלמות של אוקטבה.

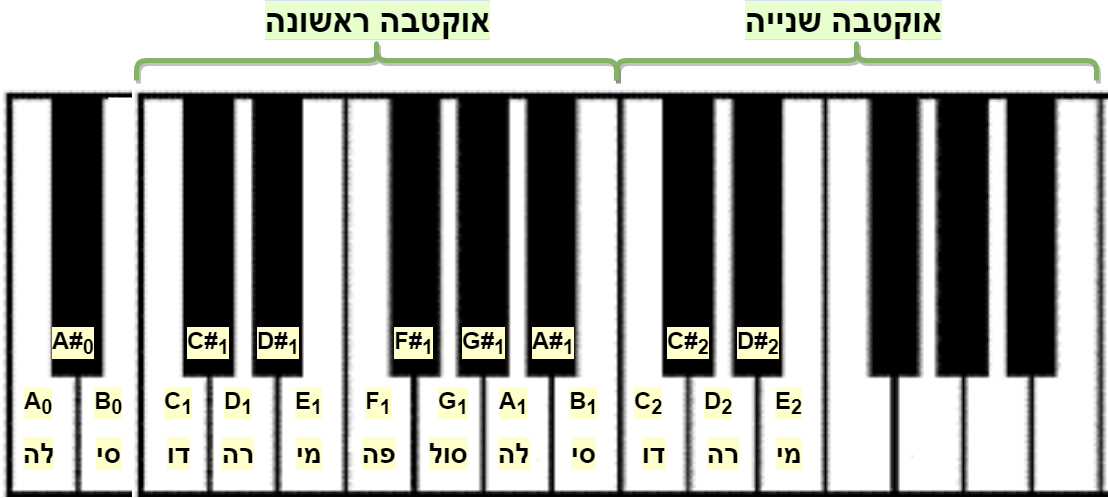
את שנים-עשר הצלילים הבסיסיים נהוג לסמן ע"י שבעת הסמלים לטיניים A, B, C, D, E, F, G (ובהתאמה בעברית: לה, סי, דו, רה, מה, פה וסול) **וסימני התק** '#' (**דיאז**) ו-'b' (**במול**) המציינים שינוי של חצי-טון כלפי מעלה או מטה, בהתאמה. לפיכך את שנים-עשר הצלילים הבסיסיים נוכל לרשום באופן שקול כך –

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

(ובהתאמה: לה, לה-דיאז, סי, דו, דו-דיאז,...,סול-דיאז או לה, סי-במול, סי, דו, רה-במול,...,לה-במול). בכדי להתייחס לאיזשהו גובה (תדר) קונקרטי של אחד משנים-עשר הצלילים הבסיסיים, מה שמכונה גם גובהו האבסולוטי, נוהגים לחלק את כלל הצלילים לסדרות של שנים-עשר צלילים עוקבים, החל מ- C(דו), ולסדרן על-פי מיקומן היחסי מנמוך לגבוה: בפסנתר סטנדרטי בעל שמונים ושמונה קלידים, סדרת שנים-עשר הצלילים המופקים ע"י שנים-עשר הקלידים התואמים המתחילים מקליד ה-C השמאלי ביותר במקלדת (שהוא גם ה-C הנמוך ביותר במקלדת) נקראת האוקטבה הראשונה. הסדרה העוקבת לה מימין נקראת האוקטבה השנייה, וכן הלאה. באופן זה, ניתן לסמן גובה-צליל קונקרטי ע"י ציון הסמל המייצג את הצליל הבסיסי, סימן התק במידה ונדרש, ואינדקס המציין את מספר האוקטבה הרלוונטי, למשל – C1 מייצג את גובה הצליל C הנמוך ביותר במקלדת הפסנתר, C2 את גובה הצליל C הגבוה ממנו באוקטבה אחת וכן הלאה, להלן המחשה של תוויות שמות/סמלי הצלילים המופקים ע"י 20 הקלידים הראשונים של מקלדת פסנתר סטנדרטית בעלת 88 קלידים –



**מנעד** הוא היקף של איזשהו טווח של צלילים. לפיכך מנעד של מקלדת פסנתר סטנדרטית הוא 88 חצאי טון.

1. **תווים**

בכדי לייצג ו/או לתאר יצירה מוסיקלית באופן גראפי על הכתב/דפוס, משתמשים בתווים. **תווים** הם סמלים המתארים גובה צליל או הפסקה (שקט) וכן את משך שהיית הצליל/הפסקה, אשר נקרא גם הערך הריתמי של הצליל/הפסקה. ברישום ניתן לציין גם תכונות הקשורות לצורה המיועדת להפקת הצליל, כמו דרגות דינאמיקה (חוזק/הדגשה) ותחושת מהירות/זרימה הכללית. התווים עצמם רשומים על גבי **חמשה**, מערכת של חמישה קווים אופקיים מקבילים.

* 1. **גובה צליל**

גובה צליל מיוצג ע"י גובה יחסי תואם בחמשה ביחס למפתח מסוים, עם סימן התק מתאים ע"פ הצורך. להלן דוגמה של רישום הצלילים החל מ-A3 (לה באוקטבה השלישית) ועד ל-G#5 (סול דיאז באוקטבה החמישית) על חמשה לפי מפתח סול[[5]](#footnote-5):



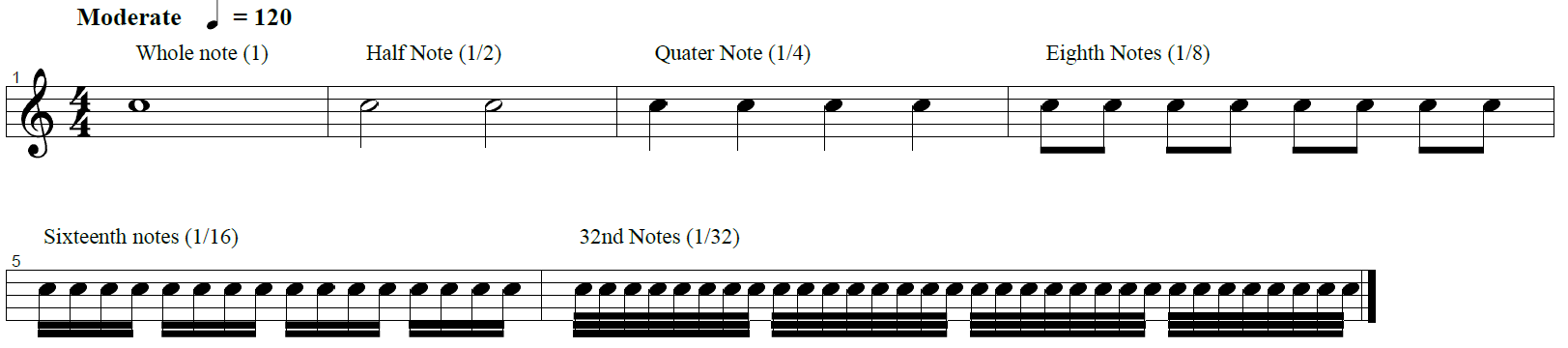
* 1. **תיבות, קצב, משקל וערך ריתמי (משך שהייה קצבי)**

התווים בחמשה מחולקים **לתיבות**, לרוב באורך קבוע, המהוות יחידות לוגיות המפרידות בין קבוצות של תווים. ניתן למספר את התיבות ולעטוף אותן בסימני חזרה והוראות בקרה המנחות לחזור עליהן 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 **BPM**[[6]](#footnote-6), המציין את מספר הפעימות שיש בדקה אחת. למשל, BPM של 120 מציין כי הקצב הוא שתי פעימות בכל שנייה.

לכל יצירה מגדירים **משקל** קצבי ע"י שבר : המכנה  מציין את מכנה יחידת הזמן הבסיסית המייצגת פעימה בודדת שהיא , והמונה  מציין כמה יחידות זמן בסיסיות כאלו יש בתיבה. למשל, המשקל הנפוץ  מציין שיחידת הזמן הבסיסית לפעימה היא רבע (ערך המכנה הוא ארבע) וכן שיש בכל תיבה ארבעה רבעים (ערך המונה הוא ארבע).

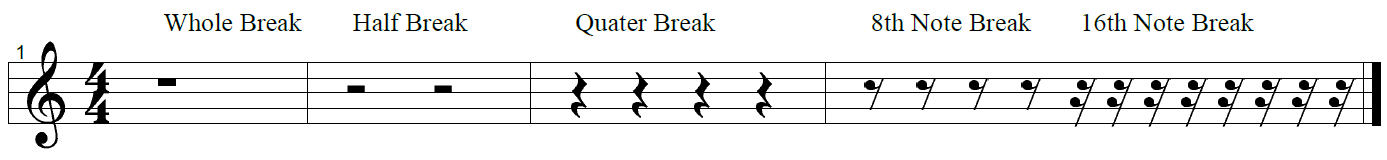
בכדי לציין את **משך השהייה** של תו נתון, מה שנקרא **ערכו** **הריתמי**, מתאימים לתו סימון קצבי המתאים לאורך השהייה שלו ביחס לתיבה שלמה. למשל, במשקל של , תו עם ערך ריתמי **שלם** מתמשך לאורך תיבה אחת שלמה (ארבעה פעימות), **חצי** מתמשך לאורך חצי תיבה (שתי פעימות), **רבע** למשך רבע תיבה (פעימה אחת בלבד), וכן הלאה. להלן המחשה של סימונים של כמה ממשכי הצליל הבסיסיים במשקל –



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

כמו כן, ניתן להאריך את משך שהיית צליל/תו אל מעבר ליחידת הזמן הבסיסית שלו ע"י רישום מופע נוסף שלו העוקב למופע המבוקש וקישור שני המופעים בקשת חיבור. הארכה שכזו קובעת את משך השהייה הכולל של התו לסכום משכי הזמנים של התווים המחוברים בקשת. לחלופין, ניתן לציין יחידת זמן מנוקדת, שמשמעה היא משך של פי 1.5 מיחידת הזמן המקורית ללא הניקוד. ע"י שימוש בשתי הטכניקות לעיל ניתן להרכיב אינסוף ערכים ריתמיים שונים, ולייצר **סינקופות**, הדגשה של איזשהו תו ע"י ניגונו שלא על הפעימה עצמה, למשל ע"י קיצור משך השהייה של התו שקודם לתו הנוכחי, וניצול הזמן שהתפנה כדי להקדים טיפה את הנגינה של התו הנוכחי, פעולה שמעוררת אפקט של הפתעה ועניין.

עבור הפסקות (שהיות של שקט) בין צלילים יש סימונים קצב ייעודיים, להלן הסימונים של הפסקות ביחידות זמן הבסיסיות מעל משקל :



1. **הרמוניה ומלודיה**

באופן כללי, **מלודיה** (מנגינה) מוגדרת ע"י רצף של צלילים המנוגנים אחד אחרי השני (לחוד), ו**הרמוניה** מוגדרת ע"י רצף של צלילים המנוגנים יחדיו (במקביל). ההרמוניה מלווה את המלודיה.

1. **סוגי מרווחים**

כאמור, מרווחים במוסיקה המערבית מרווחים נמדדים ביחידות של טון (וחצאי טון). היחס הנקבע ע"י מרווח מסוים משרה איזושהי תחושה כאשר המרווח מנוגן באופן הרמוני (דהיינו כאשר שני הצלילים היוצרים את המרווח מנוגנים יחדיו במקביל). תחושה זה יכולה להיות מסווגת כנעימה (קונסוננטית), צורמת (דיסוננטית) או זכה (צלולה, טבעית). בהתאם, המרווחים מסווגים למחלקות מתאימות. להלן רשימה של המרווחים הבסיסיים הכלולים באוקטבה אחת –

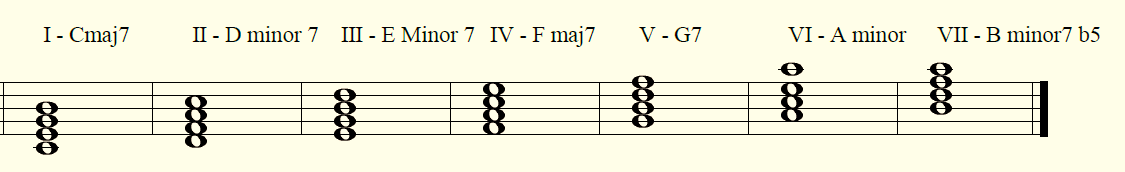
|  |  |  |
| --- | --- | --- |
| מרווח | מרחק בטונים | מחלקה |
| סקונדה קטנה | חצי טון | מרווחים דיסוננטיים |
| סקונדה גדולה | טון | מרווחים דיסוננטיים |
| טרצה קטנה | טון וחצי | מרווחים קונסוננטיים |
| טרצה גדולה | שני טונים | מרווחים קונסוננטיים |
| קוורטה | שניים וחצי טונים | מרווחים זכים |
| טריטון | שלושה טונים | מרווחים דיסוננטיים |
| קווינטה | שלושה וחצי טונים | מרווחים זכים |
| סקסטה קטנה | ארבעה טונים | מרווחים קונסוננטיים |
| סקסטה גדולה | ארבעה וחצי טונים | מרווחים קונסוננטיים |
| ספטימה קטנה | חמישה טונים | מרווחים דיסוננטיים |
| ספטימה גדולה | חמישה וחצי טונים | מרווחים דיסוננטיים |
| אוקטבה | שישה טונים (שנים-עשר חצאי טונים) | מרווחים זכים |

הערה: המרווחים נמדדים הן במרחק ביחידות של גובה הצליל האבסולוטי (חצאי טונים) והן בסימול התווי ע"י המרחק בין התווים בסולם הכרומטי, למשל המרחק האבסולוטי בין דו-דיאז לבין מי-במול הוא טון, אולם מאחר שבסולם הכרומטי של שנים-עשר הצלילים, יש בין דו למי את התו רה, המרווח הזה של טון לא יסווג כסקונדה גדולה, אלא כטרצה קטנה מוקטנת.

1. **סולמות, טונאליות ודרגות הרמוניות**

סולם הוא סדרה מחזורית של מרווחים, הקובעת איזשהו יחס סדר בין צלילים, שמרחקם זה מזה מוכתב ע"י סדרת המרווחים של הסולם. במוסיקה הטונאלית המערבית, בכל סולם שכזה יש צליל אחד שהוא "השליט" בסולם, הוא נשמע הכי טוב וטבעי בקונטקסט רצף הצלילים שבסולם: הצלילים בסולם נמצאים באינטראקציה הדדית תמידי של יצירת מתח ופתרונו ע"י רגיעה, והרגיעה או אתנחתא הגדולה ביותר שכל הרצפים בסולם שואפים אליה היא כאשר מתנגן הצליל ה-"שליט" של הסולם. צליל זה מכונה ה-**"טוניקה"**. צליל הטוניקה משתמש גם כשורש הסולם, דהיינו כצליל שממנו מתחילים את סדר רצף המרווחים, שקובע בהתאמה את הרכב הצלילים בסולם.

לאוסף הצלילים בסולם מיחסים דרגות, ע"פ מיקומם היחסי ברצף הצלילים שבסולם ביחס לטוניקה: תו הטוניקה עצמו הוא דרגה ראשונה, התו שרחוק ממנו במרחק הנקבע ע"י המרווח הראשון בסדרת המרווחים של הסולם הוא דרגה שנייה וכל הלאה. על בסיס דרגות אלו בונים **אקורדים**, רצפי שלושה תווים שונים או יותר המנוגנים יחדיו. כל סולם מכתיב אקורדים הנגזרים מהדרגות שלו כדלקמן: לכל דרגה (תו) בסולם מוסיפים את התווים שנמצאים ממנו במרחק של 3,5 ו-7, במובנים של דרגות. לאקורדים הנוצרים באופן זה קוראים **דרגות הרמוניות**. להלן דוגמה של הדרגות ההרמוניות הנגזרות מהסולם דו-מז'ור, הנקבע ע"י סדרת המרווחים: טון-טון, חצי טון, טון, טון, טון, חצי טון, שמושרש בתו דו (כלומר, סדרת המרווחים קובעת סולם מז'ור כלשהו, וכאשר משרישים סדרה זו בדו בתור צלילי הטוניקה, מתקבל הסולם דו-מזו'ר) –



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

1. Chanan Welt [↑](#footnote-ref-1)
2. שימוש ב-EntityFramework גרסה 6, לא EntityFramework Core. [↑](#footnote-ref-2)
3. ASP.NET MVC גרסה 5. גרסאות עדכניות יותר כבר כתובות ב-ASP .NET Core, גרסת ה-Cross-Platform של .NET. [↑](#footnote-ref-3)
4. Plain Old CLR (Common Language Run-time) Object [↑](#footnote-ref-4)
5. מפתח סול מציין שמיקום התו G4 (סול באוקטבה רביעית) על פני החמשה הוא השורה השנייה מלמטה בחמשה. [↑](#footnote-ref-5)
6. BPM = Beats Per Minute [↑](#footnote-ref-6)