**Generative Music Platform**

עבודת גמר 5 יח"ל במגמת מדעי המחשב



**נושא הפרוייקט –** Generative Music

**שם התלמיד –** ערן מחלב

**ת"ז –** 323103150

**שם המנחה** **–** תומר טלגם

**מורה מלווה –** ירון יצחקי

**שם בית ספר –** תיכון אהל שם

**שנה –** 2019

<https://github.com/emachlev/GenerativeMusicPlatform>

תוכן עניינים

[תקציר כולל ורקע 3](#_Toc9526718)

[שפת התכנות וסביבת העבודה 4](#_Toc9526719)

[מודולים וספריות שהשתמשתי בהם בפרוייקט 5](#_Toc9526720)

[מבנה המערכת ואופן פעולתה 7](#_Toc9526721)

[יצירת מנגינה 10](#_Toc9526722)

[הרכבת הבתים 11](#_Toc9526723)

# תקציר כולל ורקע

שמעתם פעם שיר שכל כך אהבתם שרציתם שהוא יימשך לנצח?

כולנו היינו שם: בין אם אנחנו מקשיבים למוזיקה סתם לכיף או כדי שהיא תעזור לנו להתרכז, השירים שאנחנו אוהבים תמיד קצרים מדי. לנגן אותם על repeat זה לא דבר טריוויאלי, כי חזרה מדוייקת הופכת למשעממת דיי מהר.

**Generative Music** (מוזיקה ג'נרטיבית) הוא מונח המתאר מוזיקה המתנגנת על ידי תוכנת מחשב, ועונה על הקריטריונים הבאים:

* היא צריכה להיות שונה בכל פעם שמנגנים אותה, ללא חזרות ניכרות.
* היא צריכה להתנגן כל עוד מישהו מאזין לה.

מוזיקה ג'נרטיבית מציעה הזדמנות ליהנות כל הזמן מיצירות מוזיקליות שאנחנו אוהבים ללא החסרונות של מוזיקה מוקלטת, שהיא סופית ונמשכת זמן קבוע.

העיקרון העומד במרכז הפרוייקט שלי הוא יצירת המשכים למנגינות בעזרת אלגוריתמים מורכבים. המוצר הוא פלטפורמה היוצרת ומנגנת גרסאות אינסופיות של יצירות מוזיקליות שונות, והיא נגישה דרך הדפדפן, כך שמשתמש במערכת יוכל להאזין לה בקלות ומכל מכשיר.

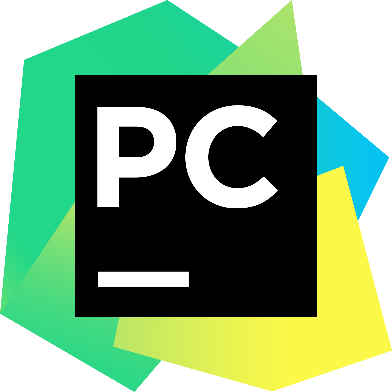
# שפת התכנות וסביבת העבודה

הפרוייקט הוא אפליקציית Web שצד השרת שלה כתוב ב-**Node.js**, סביבה המריצה קוד JavaScript מחוץ לדפדפן. Node.js מאפשרת יצירה של שרתי Web וכלים מבוססי רשת בעזרת JavaScript ואוסף של "מודולים" שמטפלים במימוש של פונקציות שונות.

ל-Node.js תמיכה רשמית ברוב מערכות ההפעלה, והקוד שלה [פתוח](https://github.com/nodejs/node) לחלוטין. ההבדל המרכזי בינה לתכנות בשפות אחרות כמו PHP הוא שרוב הפונקציות ב-PHP סינכרוניות, כלומר חוסמות את ריצת התוכנית עד שפעולתן נגמרת, בעוד פונקציות ב-Node.js הן **א-סינכרוניות**. הן רצות במקביל ומשתמשות ב-Callbacks כדי להודיע על הצלחתן או כשלונן.



השתמשתי ב-**PyCharm** בתור סביבת העבודה. היא ידועה בתור סביבת העבודה הכי נוחה לפייתון אשר מאפשרת כתיבה נוחה של הפרוייקט. גירסת ה-Pro שלה תומכת בכתיבה והרצה בשפות ופלטפורמות אחרות הכוללות את Node.js, ומספקת יכולת להוריד מודולים בלחיצה אחת ופירמוט אוטומטי של קוד.



## מודולים וספריות שהשתמשתי בהם בפרוייקט

* **Express** – framework לאפליקציות Web ו-APIים עבור Node.js. בחרתי בו כי הוא מודול מאוד דומיננטי בקהילת המפתחים ב-Node ומשתמשים בו גם באתרים פופולאריים כמו PayPal ו-Uber.



* **EJS** – מנוע templates הכתוב ב-JavaScript ואחראי על רינדור ה-HTML באפליקציה. מנועי templates מסייעים בהפרדת קוד צד השרת המיועד לעיבוד של המידע מהקוד המציג אותו למשתמש בפועל. במקום להוסיף תוכן דינאמי לאתר בקוד צד השרת, בסיום העיבוד השרת מעביר את המידע למנוע ה-templates, והוא מציב אותו במקום המיועד לכך ב-HTML. בחרתי ב-EJS כי הוא פשוט ונוח, ולא היה לי צורך במנועים עוצמתיים יותר התומכים במגוון רב של אופציות כמו ירושה של אלמנטים וטמפלייטים.
* **Material Design for Bootstrap 4** – ערכת ה-UI הטובה ביותר (לפחות לדעתי...) לבניית אתרים ואפליקציות רספונסיביים ומותאמים למובייל. MDB מציעה יותר מ-500 אלמנטים מעוצבים, אייקונים, אנימציות CSS, טמפלייטים ומדריכים לשימוש אישי ומסחרי, והכל לפי הקונבנציות של העיצוב המטריאלי של Google.



* **Tone.js** – ספרייה לניגון מוזיקה בדפדפן באמצעות ה-Web Audio API. בהינתן קבצי השמע המתאימים, Tone.js מתנהגת למעשה כפסנתר הנשלט על ידי קוד. לדוגמה:

piano.triggerAttackRelease("C3", 1);

שורה זו מדמה לחיצה על התו "דו" (יפורט בהמשך) במשך שנייה, ושחרור שלו לאחר מכן. בנוסף השמשתי בספריית tonejs-instruments שדואגת לטעינת קבצי השמע (samples) של כל התווים למחשב הלקוח.



* **jQuery** – ספריית JavaScript שמטרתה להקל על כתיבת סקריפטים לצד הלקוח. העבודה עם ספרייה זו פשוטה ונוחה, ועושה פעולות כמו הסתרת אלמנטים בדף קלות וקצרות יותר על מנת שהקוד יהיה קצר ונוח לקריאה.

JQuery logo.png

# מבנה המערכת ואופן פעולתה

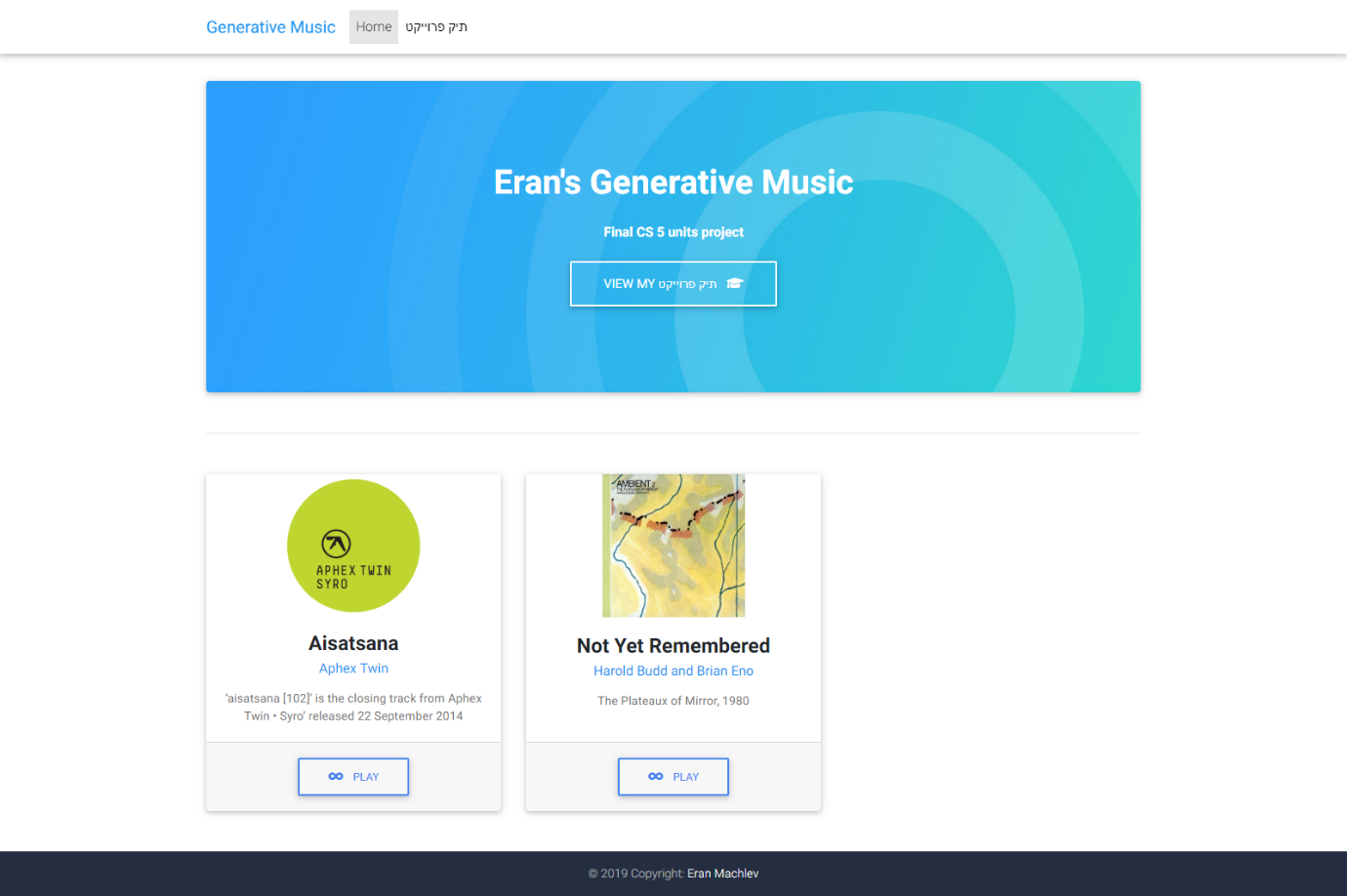
קובץ **pieces.js** – אינדקס של מנגינות מוכנות שהמערכת מציעה למשתמשים. מכיל רשימה של אובייקטים מסוג piece, לשימוש ע"י app.js (בהמשך).

* **piece** – אובייקט המייצג מנגינה ג'נרטיבית ומידע עליה. מנגינה היא מילון הכולל את המאפיינים הבאים:
  + **name** – שם היצירה שעליה מתבססת המנגינה
  + **by** – כותב היצירה
  + **description** – תיאור המנגינה
  + **file** – מיקום של סקריפט JavaScript היוצר את המנגינה (ביחס ל-/static/js/pieces)
  + **image** – מיקום של קובץ תמונה עבור המנגינה (ביחס ל-/static/img/pieces)

**const** pieces = [  
 assemblePiece('Aisatsana', 'Aphex Twin', '‘aisatsana [102]’ is the closing track from Aphex Twin • Syro’ released 22 September 2014', 'aisatsana.js', 'aisatsana.jpg'),  
 assemblePiece('Not Yet Remembered', 'Harold Budd and Brian Eno', 'The Plateaux of Mirror, 1980', 'not\_yet\_remembered.js', 'not\_yet\_remembered.jpg'),  
];  
  
**function** assemblePiece(name, by, description, file, image='') {  
 **return** {'name': name, 'by': by, 'description': description, 'file': file, 'image': image}  
}  
  
module.exports = pieces;

קובץ **app.js** – הסקריפט הראשי שמופעל עם הרצת האפליקציה. מייבא את רשימת המנגינות מ-pieces.js, קובע את המודולים והפרמטרים הדרושים לריצת השרת, ה-templating engine ואת תיקיית הקבצים הסטטיים, ומתחיל להקשיב לחיבורים מלקוחות:

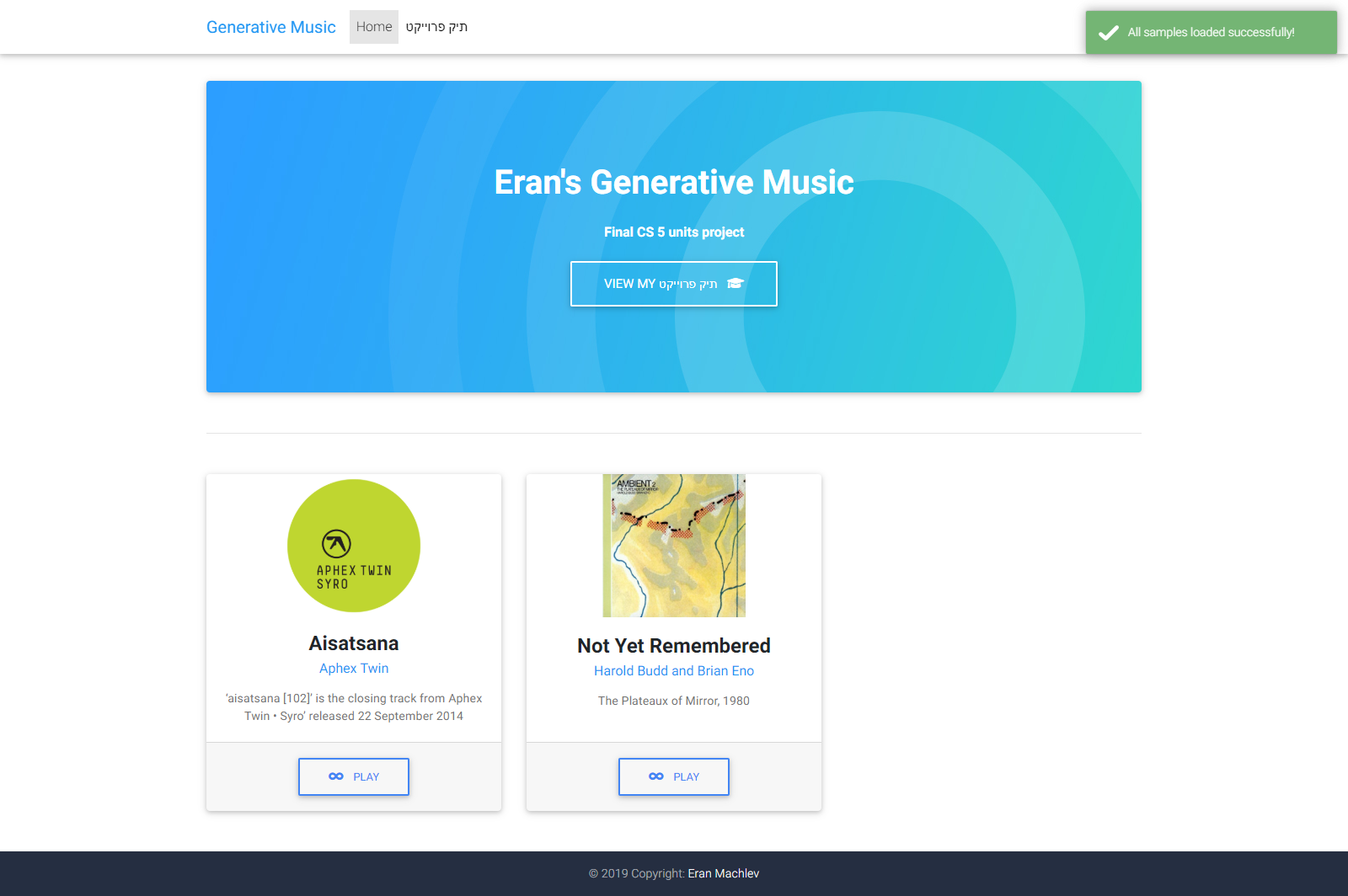
**const** pieces = require('./pieces.js');  
  
**const** PORT = 8081;  
  
// server.js  
// load the things we need  
**const** express = require('express');  
**const** app = express();  
  
// set the view engine to ejs  
app.set('view engine', 'ejs');  
// set the static path  
app.use('/static', express.static('static'));  
  
// index page  
app.get('/', **function** (req, res) {  
 res.render('pages/index', {  
 pieces: pieces  
 });  
});  
  
app.listen(PORT);  
console.log('Listening on: localhost:' + PORT);

כאשר לקוח נכנס לעמוד הראשי באתר, השרת מחזיר לו את עמוד האינדקס עם כל המנגינות שהוא יכול להקשיב להן:

קובץ **/static/js/index.js** – סקריפט הלקוח הראשי. רץ במחשב הלקוח עם טעינת האתר, ולאחר טעינת כל המודולים הדרושים לפעולת המערכת. סקריפט זה מגדיר את האובייקטים והפעולות שהמנגינות צריכות כדי לרוץ כהלכה.

**let** schedule = **null**; // The schedule of the currently playing piece  
**let** playingCaller = **null**; // The Play button that was pressed to play the currently playing piece  
  
**function** playPiece(file, caller) {  
 **if** (Tone.context.state !== "running") {  
 Tone.context.resume();  
 }  
 **let** tempPlayingCaller = playingCaller;  
 **if** (schedule) {  
 stopPiece(playingCaller)  
 }  
 **if** (caller !== tempPlayingCaller) {  
 $.getScript(file);  
 $(caller).html('<i class="fas fa-stop mr-2 piece-play-icon"></i> Stop');  
 $(caller).removeClass('btn-outline-primary');  
 $(caller).addClass('btn-primary');  
 playingCaller = caller;  
 }  
}  
  
**function** stopPiece(caller) {  
 Tone.Transport.cancel();  
 schedule = **null**;  
 $(caller).html('<i class="fas fa-infinity mr-2 piece-play-icon"></i> Play');  
 $(caller).removeClass('btn-primary');  
 $(caller).addClass('btn-outline-primary');  
 playingCaller = **null**;  
}  
  
**let** piano = SampleLibrary.load({  
 instruments: "piano"  
});  
  
Tone.Buffer.on('load', **function** () {  
 $('.piece-play-btn').removeAttr('disabled');  
 toastr.success('All samples loaded successfully!');  
 piano.toMaster();  
 Tone.Transport.start();  
});

בהתחלה, הסקריפט מאתחל את אובייקט הפסנתר ע"י טעינת ה-samples הדרושים, ולאחר מכן הוא מאפשר למשתמש לנגן כל אחת מהיצירות הזמינות באתר ומודיע לו על כך באמצעות התראת toast:



על שאר הפעולות והמשתנים בקובץ זה ארחיב בפרק הבא.

# יצירת מנגינה

פרק זה הוא החלק המרכזי בפרוייקט וכאן בא לידי ביטוי המחקר שביצעתי. המשימה הייתה לקחת יצירה מוזיקלית מוכנה, ולבנות תהליך שמנגן אותה במבנה ובחוקיות שלה בלי סוף וללא חזרות (כלומר: לחולל לה המשך התואם את מאפייניה הייחודיים). תחילה חיפשתי מנגינה מתאימה בעלת מבנה קבוע שיהיה נוח לבנות ממנה גירסה ג'נרטיבית, ובחרתי ב-[aisatsana](https://www.youtube.com/watch?v=3_MRe3JwFc8) מאת Aphex Twin. כדי להסביר למה aisatsana היא בחירה מעולה למטרה הזאת, אצטרך להגדיר כמה מושגים:

* **תו** – צליל בעל שם מוגדר, כמו "דו", "רה", "מי"...
* **פעימה** – יחידת הבסיס של הזמן במנגינה. המהירות שבה יצירה מנוגנת יכולה להימדד בפעימות לדקה, או **BPM** (קצב). הקצב של aisatsana הוא 102 BPM. זאת אומרת שאורך פעימה אחת ביצירה הזאת הוא קצת פחות מ-0.6 שניות.

ל-aisatsana מבנה פשוט יחסית, והיא עוקבת אחרי דפוס קבוע: מהתו הראשון, כל 16 פעימות מכילות סדרה עם כמות מסויימת של תווים, שאותה נכנה "בית".

לדוגמה, בין השנייה השלישית לשנייה ה-12 נמצא הבית הראשון. בין השנייה ה-12 לשנייה ה-21 נמצא הבית השני, וכך הלאה עד סוף היצירה, שכוללת 32 בתים בסך הכל.

יצירה זו פשוטה בהרבה מהשירים הפופולאריים המקובלים כיום, שמורכבים מחלקים שקיים ביניהם שוני מהותי, כמו בית ופזמון.

מנקודת המבט הזאת, למעשה ניתן לתאר את aisatsana כאלגוריתם:

* כל 16 פעימות בצע את התהליך הבא {
  + נגן בית באורך 16 פעימות
* }

מערכת שתייצר מנגינה בסגנון של aisatsana תצטרך להתבסס על האלגוריתם הנ"ל. פעימה היא יחידת זמן שניתן להמיר לשניות בהינתן BPM של השיר, אז נציג זאת בשפה שהמחשב יבין:

* **BPM** = 102
* **זמן\_פעימה** = (1/102)\*60
* כל **16)\*זמן\_פעימה( שניות** בצע את התהליך הבא {
  + נגן בית באורך **16\*זמן\_פעימה שניות**
* }

כעת אסביר על האופן בו המערכת מרכיבה בכל פעם את הבית שעליה לנגן.

## הרכבת הבתים

דרך אחת ולגמרי לגיטימית לבנות גירסה אינסופית של aisatsana היא להפריד את כל ה-32 בתים במנגינה המקורית, וכל 16 פעימות לבחור בית אקראי ולנגן אותו. התוצאה תהיה שונה מהמקור, והיא תהיה גם אינסופית, אך משתמש שמאזין לה הרבה זמן יזכור בסוף את כל 32 הבתים, וכאמור, חזרה מדוייקת נהיית משעממת דיי מהר.

**חשוב שהמערכת תדע לבנות ולנגן בתים חדשים בנוסף לבתים הקיימים ביצירה, ושהם יישמעו דומה לבתים המקוריים.**

לא ניתן להגריל תווים אקראית ולנגן אותם כל 16 פעימות. אידיאלית, מישהו שאף פעם לא שמע את המנגינה המקורית יוכל להקשיב לגירסה האינסופית שלה בלי שהוא יוכל להבדיל בין הבתים המקוריים לבתים שהמערכת מרכיבה בעצמה.

* . לדוגמה, בעמוד האינדקס השרת מכין רשימה של מנגינות שניתן לנגן באתר, ומעביר רשימה שלהן למנוע ה-templates:

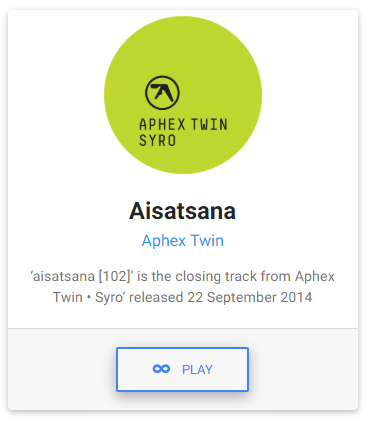
res.render('pages/index', {  
 pieces: pieces  
});

pieces היא רשימה המורכבת על ידי השרת באופן דינמי. המנוע בונה את קובץ ה-EJS הנקרא index, ומקבל כנתון את רשימת המנגינות. כל מנגינה היא אובייקט מסוג מילון, ובקובץ ה-EJS נעשית איטרציה על הרשימה:

<!--Grid row-->  
<div class="row mb-4 wow fadeIn">  
 <% pieces.forEach(**function**(piece) { %>  
 <%- include('../partials/piece.ejs', {piece: piece}); %>  
 <% }); %>  
</div>  
<!--Grid row-->

קבצי EJS הם קבצי HTML רגילים לכל דבר, רק שניתן לכתוב בהם קוד JavaScript בין תגיות ה- <% %>, דבר המקל על ההרכבה של המידע המוצג וההגשה שלו ללקוח כ-HTML רגיל. בדוגמה זו המנוע עובר על כל המנגינות, ועבור כל אחת הוא עושה include לקובץ EJS נוסף המייצג אלמנט של מנגינה, עם הפרמטרים של המנגינה הזאת. להלן התוכן של piece.ejs, עם כל המקומות המוגדרים לתוכן הדינמי:

<!--Grid column-->  
<div class="col-lg-4 col-md-12 mb-4">  
  
 <!--Card-->  
 <div class="card piece-card">  
  
 <!--Card image-->  
 <div class="view overlay">  
 <div class="rounded-top">  
 <% **if** (piece.image) { %>  
 <img src="/static/img/pieces/<%= piece.image %>" class="img-fluid piece-image ml-auto mr-auto">  
 <% } **else** { %>  
 <img src="https://mdbootstrap.com/img/Photos/Others/gradient1.jpg" class="img-fluid piece-image">  
 <% } %>  
 </div>  
 </div>  
  
 <!--Card content-->  
 <div class="card-body">  
 <!--Title-->  
 <h4 class="card-title"><strong><%= piece.name %></strong></h4>  
 <h6 class="card-subtitle blue-text pb-3"><%= piece.by %></h6>  
 <!--Text-->  
 <p class="card-text"><%= piece.description %></p>  
 </div>  
 <div class="card-footer">  
 <button onclick="playPiece('/static/js/pieces/<%= piece.file %>', **this**)"  
 class="btn btn-outline-primary piece-play-btn" disabled><i  
 class="fas fa-infinity mr-2 piece-play-icon"></i> Play  
 </button>  
 </div>  
  
 </div>  
 <!--/.Card-->  
  
</div>  
<!--Grid column-->

וזאת דוגמה לאלמנט שנוצר: