**FoodFinder**

**פרוייקט SQL - מסמך תיעוד**

**מגישים:**

**חן אילון - 201617032**

**גל קשי - 204572861**

**יוגב ניומן - 305541021**

**בן בלייר - 302678446**

תוכן

[מבוא 2](#_Toc535790577)

[מאגר הנתונים 2](#_Toc535790578)

[תרשים 2](#_Toc535790579)

[טבלאות ואינדקסים 3](#_Toc535790580)

[מבנה הDB 4](#_Toc535790581)

[הData שבמאגר הנתונים 6](#_Toc535790582)

[ספריות וחבילות חיצוניות בשימוש 7](#_Toc535790583)

[אפלקציה ושליפות SQL 7](#_Toc535790584)

[השליפות שנעשות בDB 7](#_Toc535790585)

[אפלקציה - מבנה הקוד 7](#_Toc535790586)

[ספריות וחבילות חיצוניות בשימוש 7](#_Toc535790587)

[הסבר כללי על היישום 7](#_Toc535790588)

# מבוא

"זמן הוא מה שאנחנו רוצים יותר מכל, אך גם הדבר שבוא אנו משתמשים באופן הכי גרוע" - וויליאם פן.

במציאות של היום לפעמים אין לנו מספיק זמן ללכת לקניות, או למצוא את המתכון הנכון לדיאטה שלנו.

לפעמים אנחנו צריכים מתכון מהיר עם מרכיבים שיש לנו במקרר, ולפעמים אנחנו צריכים מתכון שלא מכיל מרכיבים מסויימים, או ששייך לדיאטה כלשהי.

בשביל להקל על החיפוש יצרנו את FoodFinder, אפלקציה שמאפשרת שליפה חכמה על מתכונים.

FoodFinder מאפשרת שימוש מיטבי בזמן המשתמש, במרכיבים שנמצאים במקרר שלו ובדיאטות שלו.

האפלקציה מאפשרת שליפה על מתכון לפי זמן ההכנה שלו, מרכיבים שנמצאים או לא נמצאים בו ולפי דיאטות.

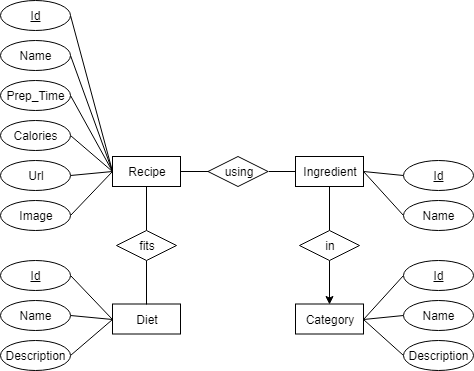
האפלקציה מאפשרת גם שליפות חכמות יותר המותאמות למצבים יומיומיים כגון:

שליפה על 3 מתכונים שניתן להכין יחד בשעתיים - למקרה שיש זמן מוגבל להכין ארוחה בת כמה מנות,

שליפה שמאפשרת לבחור מתכון לפי זמן הכנה מקסימלי, שליפת מתכון מוכר, שליפת מתכון בהתאם לדיאטה ועוד.

# מאגר הנתונים

## תרשים



## טבלאות ואינדקסים

להלן רשימת הטבלאות בDB שלנו.

סימונים:

* מפתח ראשי של טבלה יסומן ע"י קו תחתון
* אינדקס מסומן ע"י צבע ירוק
* אינדקס מסוג FULLTEXT מסומן בצבע כחול

**Tables:**

Category(Id, Name, Description)

Diet(Id, Name, Description)

Ingredient(Id, Name, Category\_Id)

Recipe(Id, Name, Prep\_Time, Calories, Url, Image)

Recipe\_Diet( Recipe\_Id, Diet\_Id)

Recipe\_Ingredient(Recipe\_Id, Ingredient\_Id, Weight, Description)

רוב השדות בטבלאות מובנים.

נסביר על השדות בטבלה Recipe:

* Id - מזהה יחודיי של המתכון
* Name - שם המתכון
* Prep\_Time - זמן ההכנה של המתכון
* Calories - מספר הקלוריות עבור המתכון וכמויות הרכיבים המוכלות בו.
* Url - קישור לאתר שמכיל את המתכון המלא
* Image - קישור לתמונה של המנה

**Foreign Keys:**

* Ingredient.Category\_Id -> Category.Id
* Recipe\_Diet.Recipe\_Id -> Recipe.Id
* Recipe\_Diet.Diet\_Id -> Diet.Id
* Recipe\_Ingredient.Recipe\_Id ->Recipe.Id
* Recipe\_Ingredient.Ingredient\_Id -> Ingredient.Id

נשים לב כי לכל מתכון, רכיב, דיאטה וקטגורית אוכל יש Id, על מנת שההצמדות יהיו קלות יותר.

כמו כן, מפורטים לעיל המפתחות הזרים שמטרותיהם להצמיד בין טבלאות.

אינדקסים נוספים שאינם מפתחות ראשיים או זרים:

* Ingredient.Name - מאפשר חיפוש קל על רכיבים במתכונים.
* Recipe.Name - מאפשר שליפת טקסט על שם מתכון. לדוגמה - מתכונים של עוגות.
* Recipe.Prep\_Time - מאפשר שליפה קלה על זמן הכנה של מתכונים.
* Diet.Name - מאפשר שליפה נוחה על טבלת הדיאטות, אם רוצים לחפש דיאטה מסוימת.
* Category.Name - מאפשר שליפה נוחה על סוגי קטגוריות של רכיבי מזון.

## מבנה הDB

נשים לב כי התלויות בDB שלנו הן:

Recipe.Id -> Recipe.Name, Recipe.Perp\_Time, Recipe.Calories, Recipe.Url, Recipe.Image

Recipe.Id -> Diet.Id

Recipe.Id -> Ingredient.Id

Recipe.Id, Ingredient.Id -> Recipe\_Ingredient .Weight, Recipe\_Ingredient.Description

Diet.Id -> Diet.Name, Diet.Description

Category.Id - > Category.Name, Category.Description

Ingredient.Id - > Ingredient.Name, Category.Id

נסתכל על אוסף השדות:

R = { Recipe.Id ,Recipe.Name, Recipe.Perp\_Time, Recipe.Calories, Recipe.Url, Recipe.Image, Diet.Id, Ingredient.Id, Recipe\_Ingredient.Weight, Recipe\_Ingredient.Description, Diet.Name, Diet.Description, Category.Id , Category.Name, Category.Description, Ingredient.Name}

קל לראות שיש הפרה של BCNF.

נפתור את הBCNF ונראה כי חלוקת הטבלאות שלנו תואמת לBCNF.

סתירה: Category.Id - > Category.Name, Category.Description

פירוק:

R1 = {Category.Id, Category.Name, Category.Description}

R2 = { Recipe.Id ,Recipe.Name, Recipe.Perp\_Time, Recipe.Calories, Recipe.Url, Recipe.Image, Diet.Id, Ingredient.Id, Recipe\_Ingredient.Weight, Recipe\_Ingredient.Description, Diet.Name, Diet.Description, Category.Id , Ingredient.Name}

R1 היא BCNF.

R2 לא, סתירה: Diet.Id - > Diet.Name, Diet.Description

פירוק:

R1 = {Category.Id, Category.Name, Category.Description}

R2 = {Diet.Id, Diet.Name, Diet.Description}

R3 = { Recipe.Id ,Recipe.Name, Recipe.Perp\_Time, Recipe.Calories, Recipe.Url, Recipe.Image, Diet.Id, Ingredient.Id, Recipe\_Ingredient.Weight, Recipe\_Ingredient.Description, Category.Id , Ingredient.Name}

R1, R2 הן BCNF.

R3 לא, סתירה: Ingredient.Id - > Ingredient.Name, Category.Id

פירוק:

R1 = {Category.Id, Category.Name, Category.Description}

R2 = {Diet.Id, Diet.Name, Diet.Description}

R3 = { Ingredient.Id , Ingredient.Name, Category.Id }

R4 = { Recipe.Id ,Recipe.Name, Recipe.Perp\_Time, Recipe.Calories, Recipe.Url, Recipe.Image, Diet.Id, Ingredient.Id, Recipe\_Ingredient.Weight, Recipe\_Ingredient.Description}

R1, R2,R3 הן BCNF.

R4 לא, סתירה:

Recipe.Id, Ingredient.Id -> Recipe\_Ingredient .Weight, Recipe\_Ingredient.Description

פירוק:

R1 = {Category.Id, Category.Name, Category.Description}

R2 = {Diet.Id, Diet.Name, Diet.Description}

R3 = {Ingredient.Id , Ingredient.Name, Category.Id }

R4 = {Recipe.Id, Ingredient.Id, Recipe\_Ingredient .Weight, Recipe\_Ingredient.Description }

R5 = {Recipe.Id ,Recipe.Name, Recipe.Perp\_Time, Recipe.Calories, Recipe.Url, Recipe.Image, Diet.Id, Ingredient.Id}

כעת כל הטבלאות הן BCNF.

נרצה לבצע מספר טיובים על מבנה הטבלאות, כדי שלא יכילו הרבה שורות מיותרות, וגם יתאימו יותר למבנה השליפות שלנו:

1. נוריד מR5 את השדה Ingredient.Id, כיוון שההצמדה שקיימת בטבלה R4 מספקת לנו מספיק מידע.
2. נחלק את R5 ל2 הטבלאות הבאות:

R5 = { Recipe.Id ,Recipe.Name, Recipe.Perp\_Time, Recipe.Calories, Recipe.Url, Recipe.Image}

R6 = { Recipe.Id , Diet.Id}

מכיוון שלכל מתכון יכולות להיות מספק דיאטות (Many to Many), אך שאר השדות הם בעלי משמעות אחת (Many to One) - עדיף לפרק את הtuples ל2 סוגים.

העיצוב הסופי מקיים BCNF.

בנוסף, העיצוב מותאם לשליפות שהאפלקציה מאפשרת - שליפות מתכונים לפי דיאטות, רכיבים וקטגוריות מזון.

על השליפות נפרט בהמשך המסמך.

## הData שבמאגר הנתונים

את המידע שהכנסנו למאגר הנתונים השגנו מ3 מקורות:

* Edamam API -

Edamam מציעים שליפות חינמיות על מאגר המתכונים שברשותם באמצעות "Get Requests" שמחזיר מידע בJSONים. מהמידע שהAPI מחזיר סיננו את המידע שמעניין אותנו וחילקנו אותו לטבלאות.

בנוסף למתכונים, שלפנו מהמאגר לעיל סוגי דיאטות.

הקשיים העיקריים שנאלצנו לעבוד איתם בAPI הזה:

* הגבלה על כמות השליפות בפרק זמן מסויים.
* פרסור של רכיבים במתכונים - הAPI מחזיר רכיבים בצורה של משפט שכולל מספרים, מידות, רכיבים ופעולות, בפורמטים משתנים ולא מוגדרים.
* TheMealDB API -

מאגר זה כולל פחות מתכונים באופן משמעותי, לכן לא השתמשנו בו לצורך זה.

מהמאגר הזה שלפנו כ-500 רכיבים של מתכונים. במאגר רכיבים זה השתמשנו על מנת לבנות אוסף שאילתות לשליפה מ-Edamam. כמו כן המרכיבים נשמרו בטבלה Ingredient.

גם השליפות מהמאגר הזה נעשות באמצעות "Get Requests" שמחזיר מידע בJSONים.

* טיובים -
* 500 המרכיבים ששלפנו לא היו מספיק מכלילים בשביל לפרסר את כל רכיבי המתכונים שהורדנו. לכן, נאלצנו "לכרות" מהרכיבים הלא מפורסרים שקיבלנו, רכיבים מפורסרים. באופן זה הגדלנו את רשימת הרכיבים לרשימה שמונה 1308 רכיבים.
* בנוסף, על מנת לאפשר שליפות מתכונים שמכילים קטגוריות של רכיבים (לדוגמה - מתכונים עם ירקות), מיינו את הרכיבים ל14 קטגוריות שונות.

סה"כ הורדנו כ-30,000 מתכונים.

(הקוד לשליפת המידע מצורף במקום הנדרש)

## ספריות וחבילות חיצוניות בשימוש

בשימוש ביצירת הDB וכריית המידע -

השתמנו בספריות פייתון הבאות:

* requests
* json
* pandas
* datetime
* time
* os
* re
* math
* sqlalchemy (create\_engine)

# אפלקציה ושליפות SQL

## השליפות שנעשות בDB

1. מתכון לפי מספר המרכיבים:

apiRecipeByNumOfIngredients()

1. מתכון לפי זמן הכנה מקסימלי:

apiRecipeByMaxPrepTime()

1. מתכון לפי סוג דיאטה:

apiRecipeByDiet()

1. מתכון לפי שם:

apiRecipeByDishName()

1. ארוחה לפי מספר מתכונים\מנות (עד 5) וזמן הכנה:

()apiMealByNumRecipiesAndTotalTime

1. רשימת מצרכים לפי רשימת מתכונים (עד 5):

apiIngredientsListByRecipiesList ()

הסבר נוסף על בחירות אינדקסים?

## אפלקציה - מבנה הקוד

האפלקציה בנויה באמצעות Python ו-django בעיקרה.

אין ממשק משתמש או admin.

מודולים חשובים:

* api.py - מרכיב שליפות SQL לפי בקשה ממשק האתר. מתחבר לDB, שולף, ומחזיר את תוצאות השליפה בJSON.
* api\_url.py - אחראי על קריאה לשאילתות הנכונות שנקראות ממשק האתר בapi שמתחבר לDB שלנו ושולף תשובות.
* views.py - מחזיר את הדף אותו אנו רוצים לראות.

האתר נכתב באמצעות קבצי html, js ו-css המצורפים לקבצי הSRC (תחת תיקיות static, templates).

יש פה משהו להוסיף?

## ספריות וחבילות חיצוניות בשימוש

ספריות Python בשימוש -

* \_\_future\_\_ ( unicode\_literals)
* textwrap
* django.http ( HttpResponse)
* django.shortcuts ( render)
* django.views.generic ( TemplateView)
* django.views.generic.base ( View)
* django.http ( JsonResponse)
* django.apps ( AppConfig)
* django.conf.urls ( include, url)
* json
* project.views ( jsonApi)

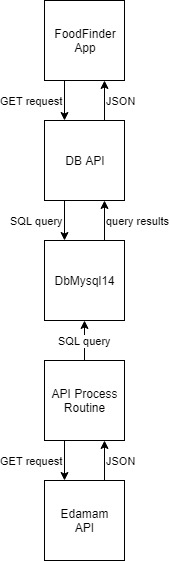
האם יש ספריות CSS ודברים אחרים?

## הסבר כללי על היישום

היישום מאפשר שליפות נוחות מהDB, בצורה קריאה למשתמש הסביר.

מדריך למשתמש מצורף בקובץ נוסף.

להלן תרשים כללי של הפרוייקט:



* שליפות ממאגרי המידע בהם השתמשנו נעשו באמצעות api של get requests"" והחזירו תשובה בצורה של JSON.
* את התשובות פרסרנו והכנסנו לטבלאות בDB שיצרנו (ומוסבר לעיל).
* על מנת להתחבר לDB, יש להשתמש בAPI שכתבנו, שמכין את השליפות, שולף ומחזיר לשואל תשובה בפורמט JSON.
* האתר מועלה לשרת של האוניברסיטה וניתן להתחבר אליו דרך האינטרנט.
* האתר מתשאל את הapi שלנו באמצעות "get request", מקבל תשובה בJSON ומציג אותה למשתמש.

האם יש מה להוסיף פה?