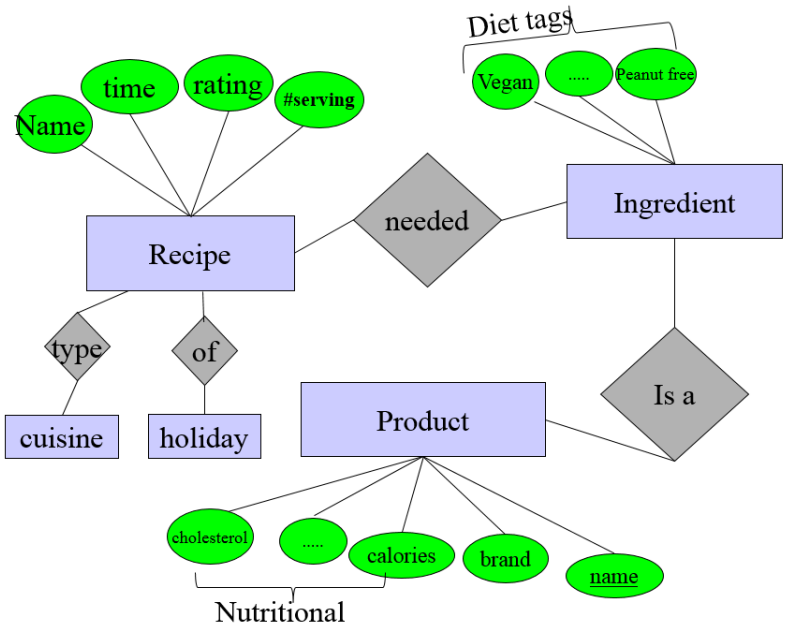
SOFTWARE DOCUMENTATION

DB SCHEME STRUCTURE

Tables:

Ingredient_tags, Products, Recipe, Recipe_Ingredient, Recipe_Cuisine, Recipe_Holiday

Entity / Relationship Diagram:



From this diagram we start our DB design, ended up with six tables:

Ingredient_tags , Products, Recipe, Recipe_Ingredient, Recipe_Cuisine, Recipe_Holiday

The basic tables are Products, Recipe, and Ingredients_tag.

They each have an additional “ID” field (not stated here), which is a unique string.

Ingredients are stored as strings in each table.

The fields that are foreign keys represent IDs in foreign tables matching their name.

In the Recipe table the recipe_name table has a full text index (for the purpose of getting, for example, “orange cake” and “chocolate cake” when searching for a cake recipe).

The Ingredients column in the table is a string for all the ingredients with portions for the recipe.

Using full text index for this would have enabled us to drop the Ingredient-In-Recipe table, but full-text index requires a lot of space (and we already have one in this table). In addition, we can save time by having the ingredient in the recipe without portions, and having connection to the recipe for later queries.

For these reason we chose to have the additional table Ingredient-In-Recipe.

Cuisine and holiday are a many-to-many relations, as they are basically like tags or categories of each recipe. We separated them from the Recipe relation to a cuisine relation and a holiday relation (that each contain only recipe id and the category in each of the tables).

The reason was that in addition to the duplicates we estimated that most of the searches would not contain these features (most of the time a user search “cake” and not cake for Christmas). Therefore, we get a smaller Recipe table and faster searching for most of the searches.

It’s important to note that in most cases the cuisine and holiday could have been a one-to-many relation; Most recipes only fit a single holiday and a single cuisine. However, there are several cases in which some of the categories intersect, such as Spring and Passover (seasons are in the holiday categories in the YUMMLY API), or South American and Argentina.

In Ingredients_tag tables each allergy or diet tag is a separate column (there are only 5 of them that we included in the database), and a binary number indicates each ingredient’s attributes. For example, milk will get 1 for vegetarian and 0 for vegan. All the columns except for the ingredient name are of type TINYINT. These things allow the table to be very compact and optimized, with no duplications of data.

However, we do understand that this is not a dynamic approach - it won’t work if we want to allow hundreds of different diets.

Also, it is wasteful for any allergies or diets that include almost all or almost none of the ingredients; For example, TREE_NUT_FREE is true for 97% of ingredients, and instead of adding a column to the table we could have added some new table with rows only for things that aren’t tree-nut free (127 rows instead of 5000 rows). Still, it would have taken much more space, since 5000 1-byte cells are still smaller than 127 100-byte rows. It would have also been a bit more complicated to do each query, since we would want to know if each diet defaults to having most products or not having most products.

Tables - Technical Description

Recipe

A recipe that can be made using ingredients, and has particular attributes that the user might care about (calorie count, number of servings, etc). Each recipe has a URL for a webpage that contains the full information about it.

PRIMARY KEY (`ID`), FULLTEXT INDEX `name` (`name`)

- ID- VARCHAR(100) NOT NULL,
- Name- VARCHAR(100) NOT NULL,
- ingredients_list- TEXT NOT NULL,
- time_needed- FLOAT NULL DEFAULT NULL,
- rating- INT(11) NOT NULL,
- number_serving- INT(11) NOT NULL,
- url- TEXT NOT NULL,
- number_of_ingredients- INT(11) NULL DEFAULT NULL,
- calories- INT(11) NULL DEFAULT NULL,
- img- TEXT NULL,

Product
Each product represents a single ingredient, and has nutritional data. These products are meant to be used as ingredients in recipes, and so end-consumer products are not meant to be here.

PRIMARY KEY (`ID`)

- ID- VARCHAR(50) NOT NULL,
- product_name- VARCHAR(100) NOT NULL,
- ingredient_name- VARCHAR(50) NOT NULL,
- brand_name- VARCHAR(50) NOT NULL,
- item_description- TEXT NULL,
- water_grams- FLOAT NULL DEFAULT NULL,
- calories- FLOAT NULL DEFAULT NULL,
- total_fat- FLOAT NULL DEFAULT NULL,
- saturated_fat- FLOAT NULL DEFAULT NULL,
- trans_fatty_acid- FLOAT NULL DEFAULT NULL,
- polyunsaturated_fat- FLOAT NULL DEFAULT NULL,
- monounsaturated_fat- FLOAT NULL DEFAULT NULL,
- cholesterol- FLOAT NULL DEFAULT NULL,
- sodium- FLOAT NULL DEFAULT NULL,
- total_carbohydrate- FLOAT NULL DEFAULT NULL,
- dietary_fiber- FLOAT NULL DEFAULT NULL,
- sugars- FLOAT NULL DEFAULT NULL,
- protein- FLOAT NULL DEFAULT NULL,
- vitamin_a_dv- FLOAT NULL DEFAULT NULL,
- vitamin_c_dv- FLOAT NULL DEFAULT NULL,
- calcium_dv- FLOAT NULL DEFAULT NULL,
- iron_dv- FLOAT NULL DEFAULT NULL,
- refuse_pct- FLOAT NULL DEFAULT NULL,
- servings_per_container- FLOAT NULL DEFAULT NULL,
- serving_size_qty- FLOAT NULL DEFAULT NULL,
- serving_size_unit- VARCHAR(50) NULL DEFAULT NULL,
- serving_weight_grams- FLOAT NULL DEFAULT NULL,

Recipe_Ingredient - many-to-many table
Represents a connection between an ingredient and a recipe: When a recipe has several ingredients there will be a row here for each of them.

PRIMARY KEY (`recipe_id`, `ingredient`), CONSTRAINT `FK_Part_of_recipe_Recipe` FOREIGN KEY (`recipe_id`) REFERENCES `Recipe` (`ID`)

- recipe_id- VARCHAR(100) NOT NULL,
- ingredient- VARCHAR(100) NOT NULL,

Ingredient_tags
A tag is a food allergy or preference. Each ingredient gets 1 if it has this tag and 0 otherwise. This means all queries are expected to filter out ingredients with tags that are 0, and not tags that are 1.

PRIMARY KEY (`ingredient_name`)

- ingredient_name VARCHAR(100) NOT NULL,
- VEGAN- TINYINT(4) NULL DEFAULT '0',
- VEGETARIAN- TINYINT(4) NULL DEFAULT '0',
- PEANUT_FREE- TINYINT(4) NULL DEFAULT '0',
- TREE_NUT_FREE- TINYINT(4) NULL DEFAULT '0',
- ALCOHOL_FREE- TINYINT(4) NULL DEFAULT '0',

Recipe_Cuisine

Many to many connection between recipe and cusine

PRIMARY KEY (`recipe_id`, `cuisine`), INDEX cuisine (`cuisine`), CONSTRAINT `FK_Recipe_Cuisine_Recipe` FOREIGN KEY (`recipe_id`) REFERENCES `Recipe` (`id`)

- recipe_id- VARCHAR(100) NOT NULL,
- cuisine- VARCHAR(100) NOT NULL,

Recipe_Holiday

Many to many connection between recipe and holiday

PRIMARY KEY (`recipe_id`, `holiday`), INDEX `holiday` (`holiday`), CONSTRAINT `FK_Recipe_Holiday_Recipe` FOREIGN KEY (`recipe_id`) REFERENCES `Recipe` (`ID`)

- recipe_id- VARCHAR(100) NOT NULL,
- holiday- VARCHAR(50) NOT NULL,

DB OPTIMIZATION

As described before, memory and time considerations were taken into account already in the DB's schema design (the separate tables for holiday and cuisine, and the Recipe_ingredients table).
For the data types in the tables, we choose the smallest possible options that fit. In the Ingredient_tags table we used in data type of TINYINT for each tag instead of string tag and many to many table. When possible, NOTNULL was chosen.

Indexes

For the indexes (or indices if you prefer), we used standard B-trees that search in O(log(n)).

There is a primary key index for each table, and also the following indexes:

Recipe:

- ID: primary key
- name: fulltext index**:** Used for optimizing queries 1, 2 and 5.

Recipe_Ingredient:

- (recipe_id, ingredient): primary key
- ingredient: key index: Used for optimizing queries 2, 3 and 5.
- recipe_id : key_index: Used for optimizing queries 1, 2, 5 and 7.

Product :

- ID: primary key
- ingredient_name: key index: Used for optimizing queries 3 and 4.

Recipe_Holiday:

- (recipe_id, holiday): primary key
- holiday: key index: Used for optimizing query 6

Recipe_Cuisine:

- (recipe_id, cuisine): primary key
- cuisine: key index: Used for optimizing query 6

Queries Optimization

Note: Yellow highlighting means that something is a parameter for the query.

1. <u>חיפוש מתכון לפי שם: (full text)</u>

בכדי לאפשר חיפוש מתכונים עם רצף מילים מסוים בשמם היה צורך לבצע חיפוש עם LIKE או חיפוש FULL TEXT. מכוון שמרבית החיפושים באתר ישתמשו בשאילתה זו החלטנו להשתמש בחיפוש FULL TEXT ויצרת FULLTEXT INDEX על השדה name ברלציה Recipe שישמש לחיפוש. כלומר העדפנו לשמור INDEX FULLTEXT למרות שהוא מצריך מקום בזיכרון כדי ליעל את החיפוש שהוא פעולה מרכזית באתר.

```
SELECT ID, name, img
FROM Recipe
WHERE MATCH (name) AGAINST (+%s IN BOOLEAN MODE)
```

<u>חיפוש מתכון לפי ID:</u>

<u>השאילתה המקורית-</u>

```
SELECT ID, name, GROUP_CONCAT(Recipe_Ingredient.ingredient SEPARATOR '\n'),
time_needed/60, rating, number_serving, url, calories, img
FROM Recipe
JOIN Recipe_Ingredient ON Recipe_Ingredient.recipe_id = Recipe.ID
WHERE ID =%s
```

מבצעים JOIN בין הרלציה Recipe לרלציה Recipe_Ingredient , בוחרים רק את הרשומה של המתכון עם ה ID המבוקש, ומחזירים את השדות הדרושים.

החיפוש מתבצע ביעילות בעזרת ה PRIMARY KEY INDEX על השדה ID ברלציה Recipe.

<u>אופטימיזציה:</u>

```
SELECT ID, name, GROUP_CONCAT(Recipe_Ingredient.ingredient SEPARATOR
'\n'), time_needed/60, rating, number_serving, url, calories, img
FROM (
SELECT ID, name, time_needed, rating, number_serving, url, calories, img
FROM Recipe
WHERE ID = %s
) as a
JOIN Recipe_Ingredient ON Recipe_Ingredient.recipe_id = a.ID
```

ראשית בוחרים מתוך הרלציה Recipe את הרשומה עם הID המבוקש, ורק לאחר מכן מבצעים JOIN עם הרלציה Recipe_Ingredients.

החיפוש ברלציה Recipe מתבצע ביעילות באמצעות הindex על השדה ID.

בנוסף, הJOIN עם הרלציה Recipe_Ingredient מתבצע רק עם רשומה אחת מכיוון שהשדה ID הוא מפתח.
יתר על כן, לרלציה Recipe_Ingredient יש index על השדה recipe_id ולכן ה JOIN יכול להתבצע ביעילות באמצעות index-nested loop join.

2. <u>חיפוש מתכון לפי אלרגיות\דיאטות:</u>

<u>השאילתה מקורית-</u>

```
SELECT DISTINCT Recipe.ID, Recipe.name, Recipe.img
FROM Recipe
WHERE MATCH (Recipe.name) AGAINST (%s IN BOOLEAN MODE)
AND NOT EXISTS (SELECT *
FROM Recipe_Ingredient
JOIN Ingredient_tags ON Ingredient_tags.ingredient_name =
Recipe_Ingredient.ingredient
WHERE Recipe_Ingredient.recipe_id = Recipe.ID
AND (Ingredient_tags.VEGAN=0
OR Ingredient_tags.VEGETARIAN=0
OR Ingredient_tags.PEANUT_FREE=0
OR Ingredient_tags.TREE_NUT_FREE=0
OR Ingredient_tags.ALCOHOL_FREE=0))
```

מחפשים מתכונים המתאימים לשם שהוכנס כפרמטר, ועבור כל מתכון בודקים שהוא לא מכיל רכיב אסור. בכל פעם מבצעים JOIN בין הטבלאות Recipe_Ingredient ו Ingredient_tags ובודקים האם קיימת רשומה עם הID של המתכון הנוכחי המכילה רכיב אסור.

<u>אופטימיזציה-</u>

```
SELECT a.ID, a.name, a.img FROM
(SELECT DISTINCT Recipe.ID, Recipe.name, Recipe.img
FROM Recipe
WHERE MATCH (Recipe.name) AGAINST (%s IN BOOLEAN MODE)) as a
LEFT join
(SELECT Recipe_Ingredient.recipe_id
FROM Recipe_Ingredient
JOIN Ingredient_tags ON Ingredient_tags.ingredient_name = Recipe_Ingredient.ingredient
WHERE (Ingredient_tags.VEGAN=0
OR Ingredient_tags.VEGETARIAN=0
OR Ingredient_tags.PEANUT_FREE=0
OR Ingredient_tags.TREE_NUT_FREE=0
OR Ingredient_tags.ALCOHOL_FREE=0)) as b
ON a.ID = b.recipe_id
WHERE b.recipe_id is NULL
```

1. בוחרים את הרשומות מRecipe לפי השם שהוכנס כפרמטר באמצעות full text index על השדה name.
2. מבצעים JOIN בין Recipe_Ingredient לבין Ingredient_tags, ובוחרים את הרשומות שמכילות רכיב אסור.
3. מבצעים LEFT JOIN בין תוצאות 1 ו 2 ובוחרים את הרשומות שמופיעות בתוצאה 1 ולא ב2.

בדרך זו, מבצעים JOIN בין Recipe_Ingredient ל Ingredient_tags פעם אחת בלבד, ולא עבור כל איטרציה כמו בשאילתה המקורית. כמו כן, קיים אינדקס על Ingredient_tags.ingredient_name ועל Recipe_Ingredient.ingredient ולכן הJOIN יכול להתבצע ביעילות.

3. <u>חיפוש כל המוצרים במתכון מסוים:</u>

<u>השאילתה מקורית-</u>

```
SELECT DISTINCT Products.product_name, Products.brand_name,
,Products.calories, Products.total_fat, Products.saturated_fat
Products.cholesterol, Products.sodium, Products.sugars, Products.protein
FROM Product
JOIN Recipe_Ingredient
ON Recipe_Ingredient.ingredient = Product.ingredient_name
WHERE recipe_id = %s
```

מבצעים JOIN בין הרלציה Products לרלציה Recipe_Ingredient לפי שם המוצר\הרכיב, ולאחר מכן בוחרים את הרשומות המתאימות ל recipe_id מסוים.

<u>אופטימיזציה-</u>

```
SELECT a.ID, a.name, a.img FROM SELECT DISTINCT Product.product_name,
Product.brand_name, Product.calories, Product.total_fat, Product.saturated_fat,
Product.cholesterol, Product.sodium, Product.sugars, Product.protein
FROM (
SELECT Recipe_Ingredient.ingredient
FROM Recipe_Ingredient
WHERE recipe_id = 'PB-_-J-Uncrustables-1246476'
) as a
JOIN Product ON a.ingredient = Product.ingredient_name
```

מבצעים ראשית בחירה של הרכיבים המתאימים למתכון לפי recipe_id ולאחר מכן מבצעים JOIN של התוצאה עם Product לפי שם המוצר\הרכיב.

הבחירה מתוך Recipe_Ingredient לפי recipe_id מתבצעת ביעילות בזכות ה index על עמודה זו.
בנוסף, מצמצמים בצורה ניכרת את מס' הרשומות מהטבלה Recipe_Ingredient עליהם מתבצעת פעולת הJOIN עם הרלציה Product.
כמו כן, כיוון שלרלציה Product יש אינדקס על השדה ingredient_name, פעולת הJOIN תוכל להתבצע ביעילות ע"י Index Nested Loop Join.

4. <u>חיפוש המוצרים עם מינימום קלוריות\שומן\כולסטרול במתכון מסוים:</u>

<u>השאילתה מקורית-</u>

```
SELECT p1.ingredient_name, p1.brand_name, p1.calories, p1.total_fat, p1.saturated_fat,
p1.cholesterol, p1.sodium, p1.sugars, p1.protein
FROM Recipe_Ingredient as a
JOIN Product as p1 ON a.ingredient = p1.ingredient_name
JOIN
(SELECT p2.ingredient_name, min(p2.calories) as min_cal
FROM Product as p2
GROUP BY p2.ingredient_name) as b ON b.ingredient_name = p1.ingredient_name
WHERE p1.calories = b.min_cal
AND recipe_id = %s
```

1. מבצעים JOIN בין Recipe_Ingredient ל Product
2. מחשבים לכל שם של מוצר את המינימום קלוריות
3. מבצעים JOIN בין 2 התוצאות, ובוחרים רשומות בהן מס' הקלוריות שווה למינימום קלוריות, וה recipe_id שווה לID המבוקש.

<u>אופטימיזציה-</u>

```
SELECT p1.ingredient_name, p1.brand_name, p1.calories, p1.total_fat, p1.saturated_fat,
p1.cholesterol, p1.sodium, p1.sugars, p1.protein
FROM (SELECT ingredient FROM Recipe_Ingredient WHERE recipe_id = %s) as a
JOIN Product as p1 ON a.ingredient = p1.ingredient_name
JOIN
(SELECT p2.ingredient_name, min(p2.total_fat) as min_fat
FROM Product as p2
GROUP BY p2.ingredient_name) as b ON b.ingredient_name = p1.ingredient_name
WHERE p1. total_fat= b.min_fat
```

1. בוחרים מתוך Recipe_Ingredient את הרכיבים שמתאימים ל recipe_id המבוקש
2. מבצעים JOIN לתוצאה של 1 עם Product
3. מחשבים לכל שם של מוצר את המינימום קלוריות
4. מבצעים JOIN בין 2 התוצאות, ובוחרים רשומות בהן מס' הקלוריות שווה למינימום קלוריות.

הבחירה ב1 מתבצעת ביעילות באמצעות ה index על recipe_id.

כמו כן, הבחירה ב-1 מצמצת באופן ניכר את הרשומות, ולכן הJOIN של התוצאה עם הרלציה Product יוכל להתבצע יותר במהירות.

כמו כן, מכיוון שקיים index על השדה ingredient_name ברלציה Product , הJOIN יוכל להתבצע ביעילות באמצעות index nested loop join.

5. א. חיפוש מתכונים שמכילים רכיב מסוים עם לכל היותר x רכיבים, ממוין לפי דירוג:

השאילתה מקורית-

```
SELECT Recipe.ID, Recipe.name, Recipe.img
FROM Recipe
JOIN Recipe_Ingredient ON Recipe_Ingredient.recipe_id = Recipe.ID
WHERE Recipe.ID IN
(
SELECT Recipe_Ingredient.recipe_id
FROM Recipe_Ingredient
WHERE Recipe_Ingredient.ingredient = "apple"
)
GROUP BY Recipe.ID
HAVING count(DISTINCT Recipe_Ingredient.ingredient) <= x
ORDER BY Recipe.rating DESC;
```

מבצעים JOIN בין הרלציה Recipe לרלציה Recipe_Ingredient לפי השדה ID של מתכון. משווים כל רשומה לתוצאת השאילתה המקוננת, ומחזירים רק את הרשומות עבורם סה"כ מס' הרכיבים לכל היותר x.

אופטימיזציה-

```
SELECT b.ID, b.name, b.img
FROM
(
SELECT recipe_id
FROM Recipe_Ingredient
WHERE ingredient = "milk"
) as a
JOIN Recipe as b ON b.ID = a.recipe_id
JOIN Recipe_Ingredient as c ON c.recipe_id = a.recipe_id
GROUP BY b.ID
HAVING count(DISTINCT c.ingredient) <= x

ORDER BY Recipe.rating DESC;
```

מבצעים ראשית בחירה של המתכונים המתאימים לפי השדה ingredient.
לאחר מכן מבצעים JOIN לרלציה Recipe לפי השדה ID של מתכון.
לתוצאה מבצעים JOIN נוסף לרלציה Recipe_Ingredient לפי השדה recipe_id.
מחזירים את הרשומות עבורם מס' הרכיבים הוא לכל היותר x.

הבחירה מתוך Recipe_Ingredient לפי ingredient מתבצעת ביעילות בזכות ה index על עמודה זו.
בצורה זו, מצמצמים בצורה ניכרת את הרשומות עליהם מתבצעת פעולת הJOIN הבאה.
פעולת הJOIN עם הרלציה Recipe לפי השדה הID יכולה להתבצע ע"י Index Nested Loop Join, כיוון שהעמודה ID היא PRIMARY KEY.
פעולת הJOIN עם Recipe_Ingredient תוכל להתבצע ביעילות גם כן, מהסיבה שקיים אינדקס על העמודה ingredient, לפיה הJOIN מתבצע.

ב. חיפוש מתכונים לפי שם שמכילים רכיב מסוים עם לכל היותר x רכיבים, ממוין לפי דירוג:

בדומה לסעיף א :

השאילתה מקורית-

```
SELECT Recipe.ID, Recipe.name, Recipe.img
FROM Recipe
JOIN Recipe_Ingredient ON Recipe_Ingredient.recipe_id = Recipe.ID
WHERE MATCH (Recipe.name) AGAINST ("cake" IN BOOLEAN MODE)) AND
 Recipe.ID IN
(
SELECT Recipe_Ingredient.recipe_id
FROM Recipe_Ingredient
WHERE Recipe_Ingredient.ingredient = "apple"
)
GROUP BY Recipe.ID
HAVING count(DISTINCT Recipe_Ingredient.ingredient) <= x
ORDER BY Recipe.rating DESC;
```

מבצעים JOIN בין הרלציה Recipe לרלציה Recipe_Ingredient לפי השדה ID של מתכון. לכל רשומה בודקים אם השם שהוכנס כפרמטר נמצא בתוך השדה Name בעזרת full text index, אם כן משווים אותה לתוצאת השאילתה המקוננת, ומחזירים רק את הרשומות עבורם סה"כ מס' הרכיבים לכל היותר x.

אופטימיזציה-

```
SELECT a.ID, a.name, a.img
FROM
(SELECT DISTINCT Recipe.ID, Recipe.name, Recipe.img, Recipe.rating
FROM Recipe
WHERE MATCH (Recipe.name) AGAINST ("cake" IN BOOLEAN MODE)) as
a
LEFT join
(
SELECT recipe_id
FROM Recipe_Ingredient
WHERE ingredient = "milk"
) as b ON  a.ID = b.recipe_id
JOIN Recipe_Ingredient as c ON c.recipe_id = b.recipe_id
GROUP BY a.ID
HAVING count(DISTINCT c.ingredient) <= 10
ORDER BY a.rating DESC;
```

1. בוחרים את הרשומות מRecipe לפי השם שהוכנס כפרמטר באמצעות full text index על השדה name.
2. מבצעים בחירה של המתכונים המתאימים לפי השדה ingredient.
3.מבצעים LEFT JOIN בין תוצאות 1 ל-2
לתוצאה מבצעים JOIN נוסף לרלציה Recipe_Ingredient לפי השדה recipe_id.
מחזירים את הרשומות עבורם מס' הרכיבים הוא לכל היותר x.

הבחירה מתוך Recipe_Ingredient לפי ingredient מתבצעת ביעילות בזכות ה index על עמודה זו. בצורה זו, ביחד עם זה שמצמצמים את כמות המתכונים כי עושים לפני את חיפוש ה-full index מצמצמים בצורה ניכרת את הרשומות עליהם מתבצעת פעולת הJOIN הבאה.
פעולת הJOIN עם Recipe_Ingredient תוכל להתבצע ביעילות גם כן, מהסיבה שקיים אינדקס על העמודה ingredient, לפיה הJOIN מתבצע.

6. <u>חיפוש מתכונים שמתאימים לחג מסוים\מטבח מסוים עם הכי מעט קלוריות:</u>

<u>השאילתה מקורית-</u>

```
SELECT DISTINCT Recipe.ID, Recipe.name, Recipe.img
FROM Recipe
JOIN Recipe_Cuisine ON Recipe_Cuisine.recipe_id = Recipe.ID
JOIN Recipe_Holiday ON Recipe_Holiday.recipe_id = Recipe.ID
JOIN
(
SELECT DISTINCT Recipe_Cuisine.cuisine, Recipe_Holiday.holiday,
min(Recipe.calories) as min_calories
FROM Recipe
JOIN Recipe_Cuisine ON Recipe_Cuisine.recipe_id = Recipe.ID
JOIN Recipe_Holiday ON Recipe_Holiday.recipe_id = Recipe.ID
GROUP BY Recipe_Cuisine.cuisine, Recipe_Holiday.holiday
) as min_calories_cuisine_hiliday
ON min_calories_cuisine_hiliday.cuisine = Recipe_Cuisine.cuisine AND
min_calories_cuisine_hiliday.holiday = Recipe_Holiday.holiday
WHERE Recipe.calories = min_calories
AND Recipe_Holiday.holiday = 'Summer'
AND Recipe_Cuisine.cuisine = 'Mexican'
```

מבצעים JOIN בין הרלציה Recipe לרלציה Recipe_Cuisine לפי השדה ID של מתכון, ואז מבוצע JOIN לרלציה Recipe_Holiday גם כן לפי השדה ID של מתכון.
לאחר מכן, מבוצע JOIN לשאילתה פנימית.
שאילתה זו מחשבת מהו מס' הקלוריות המינימלי במתכון עבור כל צמד של חג,מטבח.
לבסוף בשאילתה החיצונית, בוחרים את הרשומות אשר מתאימות לחג והמטבח שצוינו, ושמס' הקלוריות בהן שווה למס' הקלוריות המינימלי.

אופטימיזציה- עבור כל הפרמטרים

```
SELECT ID, name, img, calories
FROM
(SELECT a.holiday, b.cuisine, min(c.calories)  as min_cal
FROM(
SELECT Recipe_Holiday.recipe_id, Recipe_Holiday.holiday
FROM Recipe_Holiday
WHERE Recipe_Holiday.holiday = 'Summer'
) as a
JOIN
(SELECT Recipe_Cuisine.recipe_id, Recipe_Cuisine.cuisine
FROM Recipe_Cuisine
WHERE Recipe_Cuisine.cuisine = 'Mexican'
) as b ON a.recipe_id = b.recipe_id
JOIN Recipe as c ON c.ID = b.recipe_id
) as internal
JOIN Recipe_Cuisine ON Recipe_Cuisine.cuisine = internal.cuisine
JOIN Recipe_Holiday ON Recipe_Holiday.holiday = internal.holiday and
Recipe_Holiday.recipe_id = Recipe_Cuisine.recipe_id
JOIN Recipe ON Recipe.ID = Recipe_Holiday.recipe_id
WHERE Recipe.calories = internal.min_cal
```

מבוצע JOIN בין רלציה פנימית (internal) לבין הרלציות Recipe_Cuisine, Recipe_Holiday, ו Recipe לפי השדות המתאימים, ובוחרים את הרשומות עבורם מס' הקלוריות שווה למס' הקלוריות המינימלי ברלציה הפנימית.
ברלציה הפנימית (internal):
בוחרים מהרלציה Recipe_Holiday את הרשומות המתאימות לפי השדה holiday.
לאחר מכן, בוחרים מהרלציה Recipe_Cuisine את הרשומות המתאימות לפי השדה cuisine, ומבצעים JOIN בין 2 התוצאות.
לאחר מכן, מבצעים JOIN עם הרלציה Recipe לפי השדה ID.
מחשבים מהו מס' הקלוריות המינימלי לצמד של חג, מטבח.

חישוב הרלציה הפנימית:
הבחירה מהרלציות Recipe_Holiday ו Recipe_Cuisine מבוצע ביעילות באמצעות השימוש באינדקסים holiday ו cuisine ברלציות אלה.
בנוסף, מס' הרשומות מצטמצם כתוצאה מהבחירה, ולכן הJOIN מבוצע על מס' רשומות קטן יותר.
הJOIN עם הרלציה Recipe מבוצע עם השדה ID שהוא PRIMARY KEY ולכן מבוצע ביעילות באמצעות Index Nested Loop.
חישוב השאילתה החיצונית, מבוצע ביעילות גם כן, מכיוון שקיים אינדקס על השדות עבורם הJOIN מתבצע.

אופטימיזציה- ללא פרמטר מינימום קלוריות

```
SELECT ID, name, img
FROM (
(SELECT Recipe_Holiday.recipe_id
FROM Recipe_Holiday
WHERE Recipe_Holiday.holiday = 'Summer') as a
JOIN
(SELECT Recipe_Cuisine.recipe_id
FROM Recipe_Cuisine
WHERE Recipe_Cuisine.cuisine = 'Mexican'
) as b ON a.recipe_id = b.recipe_id
JOIN Recipe as c ON c.ID = b.recipe_id)
```

בוחרים מהרלציה Recipe_Holiday את הרשומות המתאימות לפי השדה holiday.
לאחר מכן, בוחרים מהרלציה Recipe_Cuisine את הרשומות המתאימות לפי השדה cuisine, ומבצעים JOIN בין 2 התוצאות.
לאחר מכן, מבצעים JOIN עם הרלציה Recipe לפי השדה ID.
מחשבים מהו מס' הקלוריות המינימלי לצמד של חג, מטבח.

הבחירה מהרלציות Recipe_Holiday ו Recipe_Cuisine מבוצע ביעילות באמצעות השימוש באינדקסים holiday ו cuisine ברלציות אלה.
בנוסף, מס' הרשומות מצטמצם כתוצאה מהבחירה, ולכן הJOIN מבוצע על מס' רשומות קטן יותר.
הJOIN עם הרלציה Recipe מבוצע עם השדה ID שהוא PRIMARY KEY ולכן מבוצע ביעילות באמצעות Index Nested Loop.

אופטימיזציה- עבור מינימום קלוריות ואחד מהפרמטרים מטבח או חג

```
SELECT Recipe.ID, Recipe.name, Recipe.img
FROM
(SELECT Recipe_Holiday.recipe_id FROM Recipe_Holiday WHERE
Recipe_Holiday.holiday =”summer”) as a
JOIN Recipe ON a.recipe_id = Recipe.ID
WHERE Recipe.calories =(
SELECT min(Recipe.calories) as min_calories
FROM Recipe
JOIN Recipe_Holiday ON Recipe_Holiday.recipe_id = Recipe.ID
WHERE Recipe_Holiday.holiday = “summer”
GROUP BY Recipe_Holiday.holiday)
```

הבחירה מהרלציות Recipe_Holiday או Recipe_Cuisine מבוצע ביעילות באמצעות השימוש באינדקסים holiday ו cuisine ברלציות אלה. בנוסף, מס' הרשומות מצטמצם כתוצאה מהבחירה, ולכן הJOIN עם Reicpe מבוצע על מס' רשומות קטן יותר.
הJOIN עם הרלציה Recipe מבוצע עם השדה ID שהוא PRIMARY KEY ולכן מבוצע ביעילות באמצעות Index Nested Loop.
הבחירה של המינימום פה נעשית שוב ע"י JOIN בין הרלציה Recipe_Holiday או Recipe_Cuisine עם הבחירה והרלציה Reicpe. שוב מדובר ב-JOIN יעיל בזכות האינדקס. אם לא היה ב-my_sql דרישה ל-ONLY_FULL_GROUP_BY היה יעיל יותר להשתמש ב-()MIN.

אופטימיזציה- עבור אחד מהפרמטרים מטבח או חג

```
SELECT Recipe.ID, Recipe.name, Recipe.img
FROM
(SELECT Recipe_Holiday.recipe_id FROM Recipe_Holiday WHERE
Recipe_Holiday.holiday =”summer”) as a
JOIN Recipe ON a.recipe_id = Recipe.ID
```

השאילתה היא ההתחלה של כל השאילתות האחרונות, ובאופן דומה ה-join שבה מבוצע ביעילות בזכות האנדקסים.

7. הכנסת חג ובחירת פרמטר של מנה עיקרית/ קינוח – קבלת מנה שמתאימה

לפרמטר מנה עיקרית-

```
SELECT Recipe.ID, Recipe.name, Recipe.img
FROM Recipe_Ingredient
JOIN (SELECT Recipe_Holiday.recipe_id
FROM Recipe_Holiday
WHERE Recipe_Holiday.holiday = "summer") as a ON  Recipe_Ingredient.recipe_id
= a.recipe_id
JOIN (SELECT Ingredient_tags.ingredient_name
FROM Ingredient_tags
WHERE Ingredient_tags.VEGETARIAN = 0 ) as b ON Recipe_Ingredient.ingredient =
b.ingredient_name
JOIN Recipe on Recipe.id = Recipe_Ingredient.recipe_id
```

הבחירה מהרלציה Recipe_Holiday מבוצעת ביעילות באמצעות השימוש באינדקס holiday. בנוסף, מס' הרשומות מצטמצם כתוצאה מהבחירה, ולכן הJOIN עם Reicpe_ingredient מבוצע על מס' רשומות קטן יותר.
פעולת הJOIN עם Recipe_Ingredient תוכל להתבצע ביעילות גם כן, מהסיבה שקיים אינדקס על העמודה ingredient, לפיה הJOIN מתבצע.
הJOIN עם הרלציה Recipe מבוצע עם השדה ID שהוא PRIMARY KEY ולכן מבוצע ביעילות באמצעות Index Nested Loop.

לפרמטר קינוח-

```
SELECT Recipe.ID, Recipe.name, Recipe.img
FROM Recipe
JOIN (SELECT Recipe_Holiday.recipe_id
FROM Recipe_Holiday
WHERE Recipe_Holiday.holiday = "Christmas"
) as a ON Recipe.ID=a.recipe_id
WHERE MATCH (Recipe.name) AGAINST ('cake cookie' IN BOOLEAN MODE)
```

הבחירה מהרלציה Recipe_Holiday מבוצעת ביעילות באמצעות השימוש באינדקס holiday. בנוסף, מס' הרשומות מצטמצם כתוצאה מהבחירה, ולכן הJOIN עם Reicpe_ingredient מבוצע על מס' רשומות קטן יותר.
הJOIN עם הרלציה Recipe מבוצע עם השדה ID שהוא PRIMARY KEY ולכן מבוצע ביעילות באמצעות Index Nested Loop.
החיפוש ב-WHERE מתבצע ביעילות בעזרת ה-full text index ב-Recipe.

CODE STRUCTURE:

Building the DB:

- **Create DB structure**

  CREATE-DB_SCRIPT.sql- Create the DB tables

- **Import Data from the APIs**

api_yummly.py:

Connecting to Yummly API and to mysql15 server. Calling for recipes from Yummly and Insert them into Recipe Recipe_Cusine and Recipe_Holiday tables. In addition, insert the recipe ingredients into Recipe_Ingredient table.

Function Name	Description
main	Connect to the API, search recipes and insert the data to the DB tables. The main uses in query_api() and populate_db() functions.
populate_db()	This function load the results json files from query_api(). Then its insert the recipe from the file into Recipe, Recipe_Cusine and Recipe_Holiday tables using insert_recipe() function. The function also insert the ingredient from the file into Recipe_Ingredient table using insert_ingredient() function .
query_api()	This function connect to the APi and do 10,000 calls to get 10,000 recipes and save them to json files using the get_res_json() and the get_recipe() functions.
insert_recipe(arr)	This function connecting to the DB and insert the input data into the Recipe, Recipe_Cusine and Recipe_Holiday tables. Input: arr – recipe json object
insert_ingredients(arr)	This function connecting to the DB and insert the input data into the Recipe_Ingredient table. Input: arr – recipe json object
get_res_json(start=0)	This function connect to the API and get recipes. Input: start- whice recipes to get(by the location) defult 0. Output: Json file of the recipes
get_recipe(id)	This function connect to the API and get specific recipe info about recipes from the get_res_json() output Input: id- the recipe to search in the API

api_nutritionix.py:

Connecting to nutritionix API and to mysql15 server. Calling for products that matches to the ingredients of our recipes (from Yummly), and Insert them into Product table.

Function Name	Description
main	Connect to the API, search proudcts and insert the data to the DB tables. The main uses in populate_db() function.
populate_db()	This function get the ingredients from the API using the get_all_ingredients(). For each ingredient the function search 50 matches products in the API using the get_res_json() function. Using insert_product() function each product inserted into the Proudct table. The function also save json file for each ingredient products.
get_all_ingredients()	This function connecting to the DB and get all the ingredients from the Recipe_Ingredient table. Output: The ingredients in the DB
insert_prod(arr, ing_name)	This function connecting to the DB and insert the input data into the Product table Input: arr – product json object ing_name – the ingredient that match to the product.
get_res_json(range, phrase)	This function connect to the API and get products.

api_edamam.py:

Connecting to edamam API and to mysql15 server. Calling for health labels for our DB ingredients and Insert them into Ingredients_tags table

Function Name	Description
main	Connect to the API, search health labels for the ingredients and insert the data to the DB tables. The main uses in populate_db() function.
populate_db()	This function get the ingredients from the API using the get_all_ingredients(). For function search each ingredient in the API using the get_res_json() function. Using insert_ingredient_tags() function each ingredient with the health labels inserted into the Ingredients_tags table. The function also save json file of the lables for each ingredient.

get_all_ingredients()	This function connecting to the DB and get all the ingredients from the Recipe_Ingredient table. Output: The ingredients in the DB
insert_ingredient_tags (arr, ing_name)	This function connecting to the DB and insert the input data into the Ingredients_tags table Input: arr – health label json object ing_name – the ingredient name
get_res_json(range, phrase)	This function connect to the API and get health labels

The Website:

queries.py: connecting to the DB ,run query and return the results.

The recipes queries select id, name and image of the results recipes.

Recipes queries functions	Description	Query number
searchRecipeByName(param)	The query search the recipes that contains param in the recipe name.	1
searchRecipeByID(param):	The query search the recipe with param as id	1
searchRecipeByDiet(name, vegan, vegetrian, peanutFree, treeNutFree, alchoholFree):	The query search the recipes that contains name in the recipe name and also have tags of the health label with val true.	2
searchRecipeWithIngredient(recipeName, ingName, maxIng)	The query search the recipes that contains recipeName in the recipe name and contains the ingredient ingName. If recipeName is null the search will be only on recipes that contains the ingredient ingName. If maxing is not null then the search narrow to recipes with maximum number of ingredients as maxing.	5
searchByCuisine(cuisine, minCalories)	Search recipes from the cuisine parameter. If minCalories is, true than search the recipe from the cuisine with the minimum calories.	6
searchByHoliday(holiday, minCalories)	Search recipes from the holiday parameter. If minCalories is, true than search the recipe from the holiday with the minimum calories.	6
searchByCuisineHolidayMin(cuisine, holiday)	Search recipes from the cuisine and holiday parameters.	6

searchRecipeHolidayCuisineMinCalories(cuisine, holiday, minCalories)	Search recipes from the cuisine and holiday parameters. with the minimum calories.	6
searchHolidayMain(param)	Search recipes from the holiday param that contain meet	7
searchHolidayDessert(param)	Search recipes from the holiday param that contain in the name cake or cookie	8

The proudcts queries select the ingredient_name, brand_name, calories, total_fat, saturated_fat, cholesterol, sodium, sugars, protein.

Products queries functions	Description	Query number
searchProductsInRecipe(param)	This function search all products that matches to the ingredients of the recipe with param id	3
searchProdMinCalories(param)	This function search products that matches to the ingredients of the recipe with param id, for each ingredient product with the min calories.	4
searchProdMinCholesterol(param)	This function search products that matches to the ingredients of the recipe with param id, for each ingredient product with the min cholesterol.	4
searchProdMinFat(param)	This function search products that matches to the ingredients of the recipe with param id, for each ingredient product with the min fat.	4

Server.py:

mange the server side, get user requests, call to the suitable query function from queries.py send the right html page and update the url.

Recipes queries functions	Description
Main	Run the app on the delte-tomcat server with the port
@app.route('/search_recipe_id', methods=['POST', 'GET']) def search_recipe_id():	Call to searchRecipeByID(id) and return the recipe.html that with the searchRecipeByID(id) result as data. Used when a specific recipe from the recipes search results chosen by clicking on his name. The recipe.html represent the recipe details with option to choose the products.
#search for products in specific recipe @app.route('/search_products_in_recipe', methods=['POST', 'GET']) def search_ing_product():	Call to one of the products queries function. Using the get request we match between the user choose and the function. The function return the ing_prod.html with the result of the chosen function as data. Used when the user choose to see the products from the recipe page. The ing_prod.html represent the recipe products in a table below the recipe details.
@app.route('/search', methods=['GET', 'POST']) def search():	Call to one of the recipe queries function. Using the get request we match between the user choose and the function. The function return the res.html with the result of the chosen function as data. Used when the user search for a recipe in each of the forms. The res.html represent the recipes with option the choose one of them to see more information.
@app.route('/search_diet') def search_diet():	Return the search_diet.html The form of searching by name with a diet (or without) Used when the user click on the search diet button in the top of the page.
@app.route('/search_by_ingredient') def search_by_ingredient():	Return the search_by_ingredient.html The form of searching by ingredient and name (or without) Used when the user click on the search ingredient button in the top of the page.
@app.route('/search_by_cuisine_holiday') def search_by_cuisine_holiday():	Return the search_by_cuisine_holiday.html The form of searching by cuisine holiday or both with the option of min calories recipe. Used when the user click on the search holiday cuisine button in the top of the page.

@app.route('/search_holiday_meal') def search_holiday_meal():	Return the search_holiday_meal.html The form of searching by dessert or main course for holiday. Used when the user click on the holiday meal button in the top of the page.
@app.route('/') def home_page():	Return the index.html

Templates:

base.html	General template used in all the site's pages. Contains the navbar, also includes common css files and js libraries.
base_form.html	A general form template. Used at search_holiday_meal.html, search_by_cuisine_holiday.html, search_by_ingredient.html, search_diet.html. Performing an ajax call to the server when the form is submitted by the user, and presents the results.
res.html	Template for presenting the recipes returned by any of the searches.
recipe.html	Template for presenting the full information about specific recipe. Also, performs an ajax call to the server when the user searches for products in the current recipe.
ing_prod.html	Template for presenting the products at a specific recipe. Used in recipe.html.
search_holiday_meal.html search_by_cuisine_holiday.html search_by_ingredient.html search_diet.html	Specific forms for each of the search criteria available.

APIS USED, AND DATA GENERATION

We used in Yummly API, nutritionix API, and EDANAM Nutrition Analysis API.

All the data from the API was generation using python script that search in the APIs and received json files, then parser them into the DB tables

The Yummly API used for the recipes. We populated 10,000 recipes to our DB for the Recipe table, Recipe_Ingredient, Recipe_Cusine table and Recipe_Holiday.

The Products table was populated by data from the nutritionix API, searching for 50 products for each ingredient that appears in any recipe. Later on products that not fit to their ingredient delete from the table, using SQL-query. (This because we only able to search products that contain the ingredient and not exect match, so for exemple for ice you can get ice cream).

The Ingredient_tags table were populated by data from the EDANAM Nutrition Analysis API. Searching each ingredient that appears in any recipe and using his tags. The tags object type in the json file is a list. In the python script we parser the list such that in the table each Ingredient query will get 1 for his labels and zero otherwise.

EXTERNAL PACKAGES/LIBRARIES WE HAVE USED

Flask, bootstrap, urllib, urllib2, json, MySQldb

- urllib and json have been used in order to query the apis and retrieve information.
- MySQLdb was used in order to performs operations on our db (DbMysql15)
- Flask has been used for running the server at server.py
- bootstrap has been used for styling the website

GENERAL FLOW OF THE APPLICATION