Software Documentation

DB scheme structure

טבלת places:

- בטבלה זו אנו שומרים את המידע המרכזי עליו מבוססת האפליקציה, מידע זה מכיל את כל המקומות ששלפנו מהAPI. כל המקומות נמצאים בלונדון.
- ישנם 4 סוגים - בתי מלון, ברים, מסעדות ומוזיאונים. (את סוג המקום אנו מזהים בעזרת join עם הטבלאות categories ו- places_categories).
- שדה id הוא primary key לכן הוא unique, בנוסף השתמשנו ב AUTO_INC.
- שדה google_id הוא המזהה של המקום בAPI של google והוא unique.
- שדה name - השם של המקום, שהוא גם אינדקס FULLTEXT לטובת החיפוש הטקסטואלי (ראו פירוט על השאילתה עצמה בהמשך), לכן המנוע של טבלה זו הוא myISAM.
- שדה rating - הדירוג של המקום.
- שדה vicinity - הכתובת של המקום.
- שדות latitude ו-longitude - קו הרוחב וקו האורך של המקום. בעזרת שני השדות הללו אנו יודעים לבצע חיפוש לפי מרחק.
- את השדות - google_id, latitude, longitude, rating אינדקסנו לאור השימוש הרב בהם בשאילתות האפליקציה, כאשר כמובן הבדיקות מהירות ויעילות יותר בזכות האינדקס.

	#	Name	Datatype	Length/Set	Unsign...	Allow N...	Zerofill	Default
PRIMARY KEY	1	id	INT	11	☐	☐	☐	AUTO_INCREMENT
UNIQUE	2	google_id	VARCHAR	50	☐	☐	☐	No default
FULLTEXT	3	name	VARCHAR	255	☐	☐	☐	No default
	4	rating	FLOAT		☐	☑	☐	NULL
	5	vicinity	VARCHAR	255	☐	☑	☐	NULL
	6	latitude	MEDIUMINT	9	☐	☐	☐	No default
	7	longitude	MEDIUMINT	9	☐	☐	☐	No default

טבלת categories:

- לכל מקום בטבלה place יש קטגוריה אחת לפחות אליה הוא שייך - מלון, בר, מסעדה או מוזיאון. בטבלה זו אנו שומרים את הפענוח של הקטגוריה מהאינדקס (מספר) לשם עצמו כטקסט.
- לכל קטגוריה אנחנו שומרים את האינדקס שלה ואת השם שלה.
- לכל קטגוריה יש ערך טקסטואלי יחיד.
- שדה id הוא primary key לכן הוא unique, בנוסף השתמשנו ב AUTO_INC.

	#	Name	Datatype	Length/Set	Unsign...	Allow N...	Zerofill	Default
PRIMARY KEY	1	id	INT	11	☐	☐	☐	AUTO_INCREMENT
	2	name	VARCHAR	255	☐	☐	☐	No default

טבלת places_categories:

- טבלה זו מקשרת בין place לבין הקטגוריות שלו. לכל מקום אנחנו שומרים בטבלה את האינדקס שלו ואת האינדקס של הקטגוריה שלו.
- לכל place יכול להיות מספר קטגוריות אליהן הוא שייך. למשל בית מלון שיש בו גם גלריה וגם בר וגם מסעדה. ולכן הפיצול בין הטבלאות (קשר many to many).
- האינדקס חייב להיות שייך לplace כלשהו, אבל הטבלה places היא עם מנוע myISAM ולכן (לצערנו) לא תומכת בforeign keys (יסומן כ FK מעתה והלאה).
- האינדקס של הקטגוריה חייב להיות שייך לקטגוריה כלשהי ולכן השדה category_id הוא FK לטבלת categories בה יש שדה id.
- הprimary key בטבלה זו מורכב משתי העמודות place_id ו_category_id, שכן ביחד הם מזהים באופן חח"ע את הקשר בין מקום לקטגוריה שלו (מבין מספר קטגוריות אליהן המקום אולי שייך) ואנו לא מעוניינים שאותו tuple כזה יופיע בטבלה פעמיים.

	#	Name	Datatype	Length/Set	Unsign...	Allow N...	Zerofill	Default
PRIMARY KEY	1	place_id	INT	11	☐	☐	☐	No default
FK, PRIMARY KEY	2	category_id	INT	11	☐	☐	☐	No default

טבלת choices:

- טבלה זו שומרת קומבינציות חיפוש שביצעו משתמשי האתר בעבר (כפי שמוסבר בחלק של user manual).
- בטבלה יש את השדות choice_id בתור מזהה עבור קומבינציית חיפוש ספציפית. בנוסף מופיע שדה פופולריות החיפוש, כאשר הפופולריות מתעדכנת כל פעם שמשתמש מחפש את אותה קומבינציית מקומות (כל פעם מוסיפים 1 לערך הקודם).
- בשדה popularity יש לנו אינדקס ובנוסף הערך בשדה popularity מאותחל לערך 1.
- השדה choice_id הוא primary key ולכן הוא unique, בנוסף השתמשנו ב AUTO_INC.
- השדה choice_id הוא FK של השדה choice_id בטבלה choices_places.

	#	Name	Datatype	Length/Set	Unsign...	Allow N...	Zerofill	Default
PRIMARY KEY	1	choice_id	INT	11	☐	☐	☐	AUTO_INCREMENT
idx_popularity	2	popularity	INT	11	☐	☐	☐	No default

טבלת choices_places:

- בהינתן קומבינציית חיפוש של משתמש/ים בעבר - יופיעו לנו זוגות של הchoice_id וה-place_id של כל המקומות אשר מופיעים בקומבינציה. כאשר כל place_id מקבל רשומה נפרדת ביחד עם הchoice_id של אותה קומבינציית חיפוש.
- השדה choice_id הוא FK לטבלת choices בה יש שדה choice_id גם כן.
- השדות choice_id וplace_id מהווים ביחד את הPK של הטבלה, לאור העובדה שעבור choice_id אחד יכולים להיות מספר place_id מתאימים ולהפך.

	#	Name	Datatype	Length/Set	Unsign...	Allow N...	Zerofill	Default
FK, PRIMARY KEY	1	choice_id	INT	11	☐	☐	☐	No default
PRIMARY KEY	2	place_id	INT	11	☐	☐	☐	No default

טבלת reviews:

- בטבלה זו מופיעים תגובות של משתמשים על מקומות שמופיעים בטבלת places.
- השדה place_id הוא אינדקס לטובת חיפוש מהיר של reviews רלוונטיים למקום ספציפי, לפי הid שלו. לא יכולנו לשים אותו בתור FK עם id מהטבלה places לאור העובדה שהיא עם מנוע myISAM שלא תומך בFK.
- השדה id הוא primary key לכן הוא unique, בנוסף השתמשנו ב AUTO_INC.
- בשדה author מופיע כותב התגובה.
- בשדה rating מופיע דירוג בין 0 ל5 שהמשתמש נתן.
- בשדה date מופיע התאריך בו נכתבה התגובה.
- בשדה text מופיעה התגובה עצמה שהמשתמש כתב.
- בטבלה זו אנו משתמשים כאשר משתמש באפליקציה רוצה לראות מידע נוסף לגבי place ספציפי. אנו נציג לו בנוסף למידע שמופיע בטבלה places גם את הreviews שכתבו משתמשים על המקום (אם יש כאלו).

	#	Name	Datatype	Length/Set	Unsign...	Allow N...	Zerofill	Default
PRIMARY KEY	1	id	INT	11	☐	☐	☐	AUTO_INCREMENT
	2	place_id	INT	11	☐	☐	☐	No default
	3	author	VARCHAR	255	☐	☐	☐	No default
	4	rating	FLOAT		☐	☑	☐	0
	5	date	DATE		☐	☑	☐	NULL
	6	text	TEXT		☐	☐	☐	No default

DB optimizations performed

1. בטבלאות בהן היה צורך בכדי לייעל את זמן החיפוש יצרנו FK כדי שייווצר אינדקס עבור אותו שדה וכדי לוודא שיש תאימות בין שתי הטבלאות השונות.
2. בטבלת places אנחנו מחזיקים את הgoogle_id של כל מקום. בחרנו לשמור אותו כי אנחנו משתמשים בו כדי לפנות באופן דינמי לGoogle Maps API על מנת להתעדכן בפרטים על המקומות שקיימים בdb. אנו משתמשים בו בעיקר כדי לשלוף reviews ואת הrating העדכני על כל מקום. בחרנו שלא להשתמש בו כמפתח ראשי של הטבלה שלנו (אלא יצרנו מפתח id משלנו) מכיוון שזה לא נכון מבחינת השרת להיות תלוי במזהה של שירות חיצוני בודד. כדוגמאות ליתרון בבחירה זו: אם דווקא היינו משתמשים בgoogle id כprimay key (יסומן כ-PK מעתה והלאה) בטבלה places, אז לא היינו יכולים להוסיף בקלות לאותה הטבלה מקומות משירותים אחרים שהם לא גוגל (לדוגמה, foursquare, booking וכו׳).
מעבר לזה, על ידי id פנימי השרת מצוי בפחות ״סכנה״ ממקרה בו גוגל משנים את הפורמט בו הם מגדירים מזהים ייחודים למקומות. לבסוף, באופן כללי זה practice נכון למנוע תלויות בשירותים חיצוניים כשמתכננים שרת.
3. בטבלה places יש לנו FULLTEXT index על השדה name, על מנת לאפשר חיפוש טקסטואלי בזמן יעיל. כמו כן, המנוע של הטבלה הינו myISAM על מנת לאפשר שימוש ב-Match against עם גרסת הMySQL שמותקנת על שרתי האוניברסיטה.
4. מעבר לטבלת places שמוסבר עליה בסעיף 3, שאר הטבלאות במסד הן עם מנוע InnoDB משום שיש שימוש נרחב בFK במסד שלנו. כמו כן יש אינדוקס מיוחד לPK בInnoDB.
בנוסף כפי שהוסבר בכיתה, מנוע זה הוא ACID.
5. בטבלה places שהיא המרכזית וקיימת ברוב השאילתות אם לא בכולן, יש לנו אינדקסים על השדות - google_id, latitude, longitude, rating לטובת בדיקות מהירות ויעילות.
6. על אף שבצורתם הטבעית קואורדינטות רוחב ואורך הם float, גילינו שאינדוקס לפי float הוא לא יעיל במיוחד. ולכן חילקנו את כל קווי הרוחב והאורך ב10,000 כדי שיהיו integer (ואוחסנו כ-MediumInt מכיוון שאין צורך ב-int שלם).
מעבר לכך, שכל קווי הרוחב מתחילים ב"51" (כי כולם עוברים בלונדון). כדי שזה יהיה עוד יותר יעיל הפחתנו מכולם 51 - כך גם נשמור פחות במסד נתונים וגם האינדוקס יעבוד מהר יותר כי כל ספרה תהיה בעלת משמעות גדולה יותר.
7. בכדי לחסוך ביעילות מקום, את הטבלה ששומרת לכל מקום את הקטגוריות שלו פיצלנו לשתי טבלאות categories ו- places_categories. ב-places_categories אנו שומרים לכל place id (שהוא FK של places.id) את ה-category ids אליהם המקום שייך. לכן, כאשר צריך לפענח מבחינת ערך טקסטואלי את הקטגוריה נפנה רק אז לטבלה categories.
ראשית, בצורה הזו אנו חוסכים במקום, כי שמירת הcategory name (שהוא VARCHAR, אשר דורש יותר מקום מהint) עבור כל place תדרוש יותר מקום.
שנית, מבחינת ארכיטקטורת שרת-לקוח, נכון יותר שהלקוח יכיר את ה״שמות״ של הקטגוריות (hote, resturant, museum, bar) ולא את הייצוג הפנימי בשרת כint. זה יכול לאפשר בעתיד לשנות את ה-ids מסוג אחד לאחר (ולבצע migration של הערכים מint לכל דבר אחר) ללא צורך בפיתוח נוסף בשרת.

8. בטבלה reviews שמנו אינדקס על השדה place_id כדי שבהינתן מקום נוכל למצוא בזמן מהיר ככל האפשר את הreviews שרלוונטיים אליו בטבלה.

Description of the queries

שאילתות מורכבות (לפי שם הפונ' שמריצה את השאילתה) -

1) `search_places_near_location`
השאילתה מחזירה את כל המקומות מסביב למיקום מסוים ומקטגוריה שבחר המשתמש, במרחק מהמיקום שנקבע ע"י המשתמש או ברירת המחדל (1.5 ק"מ), ממויינים בסדר יורד לפי המרחק מהמיקום שבחר המשתמש. בשאילתה זו אנו גם משתמשים בקונספט של paging כך שאנו מראים למשתמש 10 שורות בכל פעם מכלל התוצאות שחזרו.
אופטימיזציה : category_id בטבלה places_categories הוא FK לשדה id בטבלה categories.
השדות latitude, longitude עם אינדקס.

2) `search_places_by_name`
שאילתה זו מבצעת חיפוש FULLTEXT על השדה name בטבלה places. בנוסף, אנו רוצים קטגוריה ספציפית של מקום. את התוצאות אנחנו מחזירים במקבצים של 10, כאשר כל פעם השאילתה מקבלת offset ממנו היא מתחילה לשלוף תוצאות. כך אנו בעצם נותנים למשתמש את כל התוצאות ממוינות בסדר יורד לפי הרלוונטיות (מנגנון match against מתאים מספר בין 0 ל1 לכל תוצאה).
השאילתה מבוססת על כך שהטבלה places היא עם מנוע myISAM והשדה name יש לו אינדקס של FULLTEXT.
הערה : עבור מילים שמופיעות ביותר מ50% מהתוצאות השאילתה מתעלמת מתוך הנחה שאם המילה מופיעה ביותר מדי תוצאות לא לפיה רצינו לחפש. בנוסף בגלל הגדרות ברירת מחדל של mySQL בשרתי האוניברסיטה, אורך מילה מינימלי צריך להיות 4 תווים לכל הפחות.
אופטימיזציה : השאילתה מבוססת על מנגנון match against ולא על מנגנון LIKE. בנוסף, כאמור השדה name מאונדקס לטובת חיפוש Fulltext . השדה categories_id הוא FK של השדה id בטבלה categories. בטבלה places_categories השדה Id הוא PK ולכן גם יש לו אינדקס.

3) `get_categories_statistics`
שאילתה זו מקבלת מיקום של מקום מהטבלה places ואת הקטגוריה של". ומרחק מסוים מסביב למיקום הנקבע לפי ברירת המחדל (1.5 ק"מ) או לפי בחירת המשתמש. השאילתה תחזיר לנו כמה מקומות קיימים מכל קטגוריה (שאינה הקטגוריה של המקום!) ומה הדירוג הממוצע של כל המקומות יחד.
אופטימיזציה : לכל השדות latitude, id, longitude, rating יש אינדקס. בנוסף category_id בטבלה places_categories הוא FK לשדה id בטבלה categories. השדה places_id כאמור לא יכול להיות FK כי הטבלה places עם מנוע myISAM.

4) `get_popular_places_for_category`
השאילתה מחזירה לנו את שמות 5 המקומות מקטגוריה מסוימת, שנבחרו הכי הרבה ע"י המשתמשים (אם משתמש בוחר ללכת למקום מסוים, אותו מקום מקבל ערך גבוה ב1 מהערך

הקודם שלו לפופולריות חיפוש שלו). בנוסף ע"י חישוב שמוסבר בתיעוד הפונקציה השאילתה נותנת ציון בסקלה בין 1 ל5 לכל תוצאה כזו.
אופטימיזציה : בטבלה places_categories השדה Id הוא PK ולכן גם יש לו אינדקס, כמו כן השדה categories_id הוא FK של השדה id בטבלה categories.

5) `Get_popular_choices`
השאילתה מחזירה לנו את כל המקומות ששייכים ל-5 הקומבינציות של בחירה שהן הכי פופולריות. קרי, בהינתן 5 הקומבינציות הפופולריות ביותר שחופשו ע"י משתמשי האפליקציה השאילתא תחזיר את פרטים המקומות ששייכים לפחות לאחת מ5 הקומבינציות שנבחרו.
אופטימיזציה : בטבלה places_categories השדה Id הוא PK ולכן גם יש לו אינדקס, כמו כן השדה categories_id הוא FK של השדה id בטבלה categories. יש לנו אינדקס על השדה Popularity עליו מבוצעת הsubquery הראשונה. הsubquery הפנימי השני שולף רק את 5 התוצאות עם הפופולריות הגדולה ביותר ולכן הJOIN אח"כ עובדים מול טבלה קטנה פי כמה.

6) `lookup_choice_by_places_set`
השאילתה מקבלת רשימת id של מקומות מהטבלה places ומחזירה את ה-choice_id המתאים להם (אם קיים) מהטבלה choices אשר שייכים אליו <u>אך ורק</u> כל המקומות ששייכים לרשימה הנ"ל. מכיוון ש-choice_id הוא PK (כלומר, כל חיפוש הוא ייחודי) תוחזר תוצאה יחידה או לא בכלל.
אופטימיזציה : השדה place_id בטבלה Places הוא PK ולכן יש לו אינדקס. בנוסף השתמשנו בפונקציות SQL מתאימות (כמו GROUP_CONCAT) אשר למשל חסכו מאיתנו INNER JOIN משולש בין הטבלה choices_places לעצמה, ומאפשרות ל-workbench לבצע אופטימיזציות מאחורי הקלעים.

7) `crawl_by_location_highest_rating`
השאילתה מקבלת מיקום במפה ואזור שנקבע ע"י ברירת מחדל (ריבוע שמוגדר ע"י חצי ק"מ לכל כיוון) ומחזירה קומבינציה של מלון, מסעדה ובר, כאשר כל מקום שלקחנו הוא עם רייטינג מקסימלי. המלון יימצא ברדיוס של חצי ק"מ המיקום הנבחר, המסעדה תצמא ברדיוס של חצי ק"מ המלון והבר יימצא ברדיוס של חצי ק"מ המסעדה. בנוסף, השאילתה מוודאת שהמקומות הם זרים (דהיינו, לכל מקום יש id שונה משל השניים האחרים).
אופטימיזציה : מכל תת שאילתה מוחזרת תוצאה אחת בודדת (זו עם הרייטינג הכי גבוה), כך ל-JOIN יש תוצאה אחת בלבד לעבוד מולה. מעבר לכך, לכל השדות latitude, id, longitude, rating יש אינדקס. השדה place_id בטבלה places_categories לא יכול להיות FK כיוון שהמנוע בטבלה places הינו myISAM.

<u>שאילתות פשוטות יותר (לפי שם הפונ' שמריצה את השאילתא) -</u>

1) `get_place_by_place_id`
השאילתה מחזירה את כל השדות שקיימים בטבלה places עבור המקום שה-id בטבלה places זהה לארגומנט שהפונקציה מקבלת. בנוסף מוחזרת הקטגוריה של אותו המקום, שאותה מוצאים ע"י Join עם הטבלאות categories & places_categories בהן מוצאים את הקטגוריה המתאימה.
אופטימיזציה : id הוא PK ולכן יש עליו אינדקס.

2) `insert_new_choice`
השאילתה מקבלת רשימה של מספרי id של מקומות מהטבלה places. הפונקציה מריצה שאילתה אשר יוצרת רשומה חדשה בטבלה choices. כאשר הערך ברירת מחדל של popularity הוא 1 וה-choice_id מוגדר בתור AUTO_INC ונקבע ע"י הוספת אחד לערך האחרון של choice_id. לאחר מכן הפונקציה ניגשת ל-choice_id של השורה האחרונה בטבלה (דהיינו השורה שהרגע הוספנו) ומשתמשת ב-choice_id בכדי להוסיף לטבלה choices_places את הרשומות החדשות. כאשר הרשומות החדשות הן מהצורה - ה-choice_id הנ"ל והid של המקום הנוכחי ברשימה (ברשימה יש מספר מקומות וניצור רשומה עבור כל אחד מהם).
אופטימיזציה : choice_id בשתי הטבלאות המדוברות הוא PK ולכן יש עליו אינדקס. בטבלה choices_places השדה choice_id הוא FK של השדה choice_id בטבלה choices.

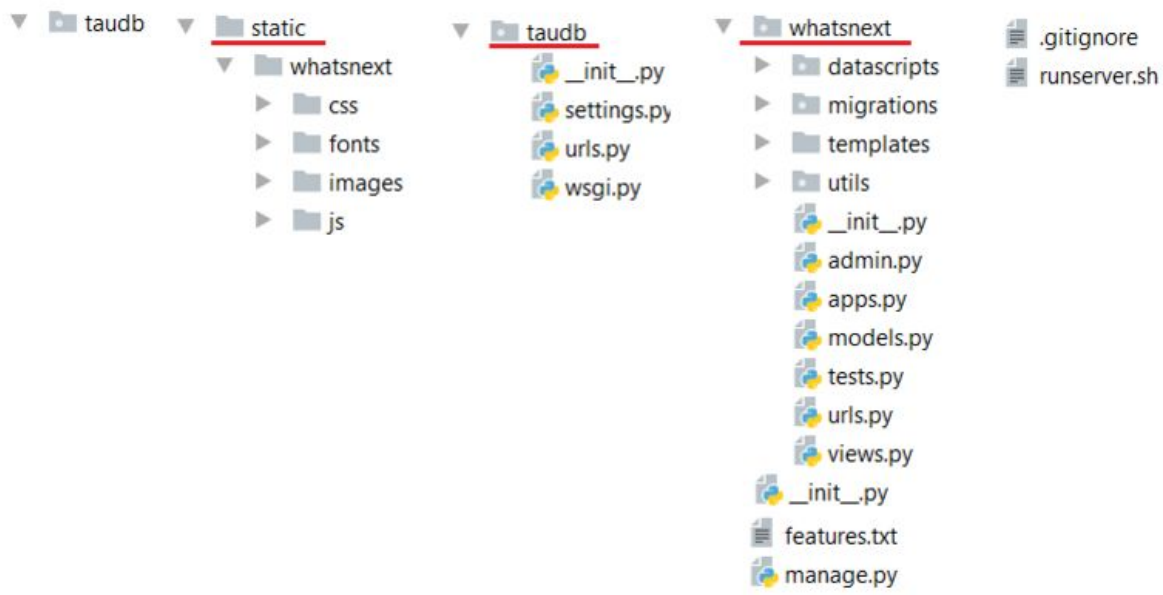
3) `update_choice`
שאילתה זו בהינתן choice_id מסוים, מעדכנת את השדה popularity בטבלה choices להיות הערך הקודם שלו ועוד 1 (קרי, מגדילה ב1).
אופטימיזציה : choice_id הוא PK ולכן יש עליו אינדקס.

4) `get_place_reviews`
שאילתה זו בהינתן id של מקום בטבלה places שולפת לו את כל הreviews שקיימים למקום בטבלת reviews.
אופטימיזציה : לשדות id בשתי הטבלאות יש אינדקס. והשדה place_id בטבלה reviews הוא FK של השדה id בטבלה places.

4) `insert_new_reviews`
שאילתא זו מכניסה review חדש לטבלה reviews.

5) `update_place_rating`
שאילתה זו מעדכנת את השדה rating של מקום ספציפי שנמצא בטבלה places.
אופטימיזציה : השדה id בטבלה places הוא PK ולכן יש לו אינדקס.

Code structure



Static :

בתיקייה זו יש לנו את צד הלקוח,קרי את התמונות בהן אנו משתמשים(למשל של הסמנים במפה), הפונטים בהם אנו משתמשים, קבצי הCSS של העיצוב וקבצי הJS שמנהלים את צד הלקוח.

Taudb :

בתיקייה זו יש לנו את החלק של הdjango, וגם את הגדרות האפליקציה (setting.py).

Whatsnext :

Datascripts :

הסקריפטים של משיכת המידע מה-API.
קואורדינטות של התחנות אוטובוס בלונדון.
קואורדינטות לפני המרה (בתור float).
קואורדינטות אחרי ההמרה (בתור mediumint).

Migrations :

השאילתות של הסכמה, של יצירת הטבלאות - scheme_queries.

Templates :

עמוד הhtml של הindex, אשר נמצא במסך המרכזי של האפליקציה.

Utils :

רוב שאילתות הSQL הן של צד השרת.

ניהול שגיאות.

קובץ python של geo_utils שעוזר להמיר מרחקים בין קווי רוחב/אורך לק"מ. ועוד פעולות.

קובץ python שמנהל את הגישה לgoogle maps API.

הקישורים בתוך האפליקציה.

קובץ views שמנהל את המידע שעובר מצד שרת לצד לקוח מבחינת השאילתות.

Description of the API use

המקומות places :

בחרנו לעבוד מול google maps API.
התמקדנו רק בעיר לונדון.
את הקואורדינטות מצאנו ע"י רשימה של קואורדינטות של רוחב ואורך של תחנות אוטובוס בעיר לונדון.
לאחר מכן משכנו מה-API מקומות שנמצאים בטווח מצומצם מסביב לאותן תחנות אוטובוס וכך קיבלנו מקומות רק מהעיר לונדון.
המקומות הם מהקטגוריות של מלונות, ברים, מסעדות ומוזיאונים (חלק מהמקומות שייכים למספר קטגוריות).
לכל מקום שמרנו מספר מאפיינים -
Id, google_id, rating, name, vicinity, longitude, latitude, category

הביקורות reviews :

את הביקורות משכנו מgoogle maps API גם כן. בהינתן place_id פנינו לAPI בבקשה לקבל ביקורות אם קיימות כאלו בAPI. כחלק מתהליך העדכון אנו מעדכנים גם את השדה rating בטבלה places בהתבסס על הביקורות החדשות.
לכל ביקורת שמרנו מספר מאפיינים -
Place_id, rating, author, date, text
כל פעם שהמשתמש לוחץ על סמן של מקום על מנת לקבל יותר פרטים אנו מבצעים חיפוש מול ה-API על מנת להוריד ביקורות חדשות אם קיימות.

External packages/libraries

Server side :

1. Django
2. MySqlDB

Client side :

1. Google maps
2. jquery

General flow of the application