מה זה בעצם JobMiner?

שימוש ב-JobMiner מאפשר חיפוש קל בפוסטים מתוך קבוצות פייסבוק המפרסמות הצעות עבודה.
בזמן שפייסבוק נעשה אחת הפלטפורמות העיקריות לחיפוש עבודה, החיפוש של עבודות בפייסבוק הינו מסורבל בגלל הרבה מגבלות:

- רוב הקבוצות ממוקדות לנישה ספציפית למשל "משרות להייטק בסן פרנסיסקו" ולא ניתן לדעת מה הם הקבוצות הכי רלוונטיות לך לחיפוש עבודה.
- גם בתוך קבוצות רלוונטיות יש הרבה פוסטים שאינם קשורים, וקשה לחפש בהם תוכן ספציפי
- יש הרבה קבוצות ובשביל לחפש בכולן צריך לעבור אחת אחת.

האתר JobMiner נותן לחפש פוסטים רלוונטים, על פי 4 קרטריונים חשובים - עיר, מדינה, חברה, סוג משרה, ובנוסף גם חיפוש על פי מילות מפתח.

***הערה: האתר מותאם כרגע לדפדפן Chrome**

תרשים הזרימה של האתר

האתר מרענן את הפוסטים שהוא שואב בצורה אוטומטית מדי יום.
כאשר נכנס משתמש הוא מזין שאילתת חיפוש רלוונטית למשרה אותה הוא מחפש.
מתוך ה-DB מוחזרים הפוסטים הרלוונטיים.
כעת המשתמש יכול לבחור:

- על ידי לחיצה על הפוסט הרלוונטי הוא יכול להיכנס לפוסט עצמו בפייסבוק ולחפש מידע נוסף
- על ידי לחיצה על הקבוצה ניתן לעבור לקבוצת הפייסבוק הרלוונטית בה הפוסט הופיע.
- על ידי לחיצה על המייל נפתח חלון שליחת מייל והמשתמש יכול לשלוח מייל למפרסם ההודעה (אם מצויין בפוסט)

תיאור מבנה בסיס הנתונים

Groups(group_id, group_fb_id, group_name, update_date) – טבלת הקבוצות	
group_id	מס' מזהה של הקבוצה, משמש *כמפתח ראשי*
group_fb_id	מס' מזהה של הקבוצה באתר Facebook
group_name	שם הקבוצה
update_date	תאריך שבו פורסם הפוסט האחרון בקבוצה
JobPost(post_id, post_story_id, publish_date, employment_form, working_manner, email, full_post_body) – טבלת הפוסטים	
post_id	מס' מזהה של הפוסט בטבלה, משמש *כמפתח ראשי*
post_story_id	מס' מזהה של הפוסט באתר Facebook
publish_date	תאריך פרסום הפוסט בקבוצה
employment_form	סוג משרה
working_manner	אופן העבודה
email	דואר אלקטרוני
full_post_body	תוכן הפוסט, *מאונדקס* לצורך חיפוש יעיל בטבלה
Companies(company_id, company_name) – טבלת החברות	
company_id	מס' מזהה של החברה, משמש כמפתח ראשי
company_name	שם החברה, משמש כאינדקס לצורך חיפוש יעיל בטבלה
CitiesStates(*city_id*, *state_id*) – טבלה מקשרת בין ערים למדינות	
city_id	מס' מזהה של העיר
state_id	מס' מזהה של המדינה

Cities(city_id, city_name) – טבלת הערים	
city_id	מס' מזהה של העיר, משמש *כמפתח ראשי*
city_name	שם העיר, משמש *כאינדקס* לצורך חיפוש יעיל בטבלה
States(state_id, state_name) – טבלת המדינות	
state_id	מס' מזהה של המדינות, משמש *כמפתח ראשי*
state_name	שם המדינות, משמש *כאינדקס* לצורך חיפוש יעיל בטבלה
JobPostGroup(*post_id*, *group_id*) – טבלה מקשרת בין פוסטים לקבוצות	
post_id	מס' מזהה של הפוסט
group_id	מס' מזהה של הקבוצה
JobPostCity(*post_id*, *city_id*) – טבלה מקשרת בין פוסטים לערים	
post_id	מס' מזהה של הפוסט
city_id	מס' מזהה של העיר

JobPostCompany(*post_id*, *company_id*) – טבלה מקשרת בין פוסטים לערים	
post_id	מס' מזהה של הפוסט
company_id	מס' מזהה של החברה
JobPostState(*post_id*, *state_id*) – טבלה מקשרת בין פוסטים למדינות	
post_id	מס' מזהה של הפוסט
state_id	מס' מזהה של המדינה

תכנון בסיס הנתונים:

בפרויקט שלנו, היחס בין פוסט לבין עיר/מדינה/חברה הוא many-to-one. כלומר, לכל פוסט יכולה להתאים רק עיר אחת, רק מדינה אחת, ורק חברה אחת (כך הוחלט שה-Parser יעבוד). מהסיבה הזו חילקנו את המידע למספר טבלאות (במקום לאכלס את כל המידע בטבלה אחת).

אפשרויות לשיפור:
אם היתה לנו האפשרות להרחיב ולשפר את הפרויקט, היינו מעוניינים לייצר יחס של many-to-many בין הפוסטים לבין ערים/מדינות/חברות. כמו כן היינו רוצים להשתמש בבינה מלאכותית וב-crowd-sourcing ע"מ לשפר את ביצועי ה Parser. ויתרנו על כך לעת עתה, בין השאר בשל הדרישה ליצור בסיס נתונים גדול – השיפורים הנ"ל גורמים לריצת ה-Parser להיות איטית בפקטור משמעותי, לכן תהליכי אכלוס בסיס נתונים בגודל המבוקש ידרוש הרבה יותר זמן.

האופטימיזציות שבוצעו

הוגדרו אינדקסים מסוג B-Tree על השדות הטקסטואליים בטבלאות הסטטיות ע"מ לאפשר חיפוש יעיל לפי השמות של ערים, מדינות וחברות – שדות city_name, state_name ו-company_name בהתאמה.

בנוסף הוגדר אינדקס מסוג FULLTEXT על השדה full_post_body בטבלה JobPost ע"מ לאפשר חיפוש יעיל בעזרת MATCH() AGAINST())נציין שהגרסא שמותקנת בשרת בפקולטה אינה מאפשרת הגדרת אינדקס כ FULLTEXT -ולכן שינינו את ה-ENGINE ל (MyISAM

תחזוקת בסיס הנתונים

הקובץ updateTimer.py מריץ עדכון כל 24 שעות. כל עדכון עובר על פוסטים שנוצרו ב 48 השעות האחרונות (עבור בניית בסיס הנתונים ההתחלתי שאבנו פוסטים שנוצרו ב 1500 הימים האחרונים).

החלק המכיל את קוד עדכון ה DB מחולק לשלושה חלקים:

- ***getPostsFromGroups*** - האחראי על איטרציה על כל קבוצות הפייסבוק ששמורות בבסיס הנתונים, ובידוד פוסט יחיד.
- ***Updater*** - האחראי על עדכון פוסט יחיד בטבלאות בבסיס הנתונים.
- ***Parser*** - האחראי על סריקת ההודעה עצמה שבפוסט, וזיהוי אלמנטים מעניינים.

getPostsFromGroups

מודול זה מעדכן את בסיס הנתונים. המודול משתמש ב-long-lived-access-token, אותו השגנו באמצעות Facebook Graph API.
startUpdate() (הפונקציה הראשית) עוברת על הקבוצות שמופיעות בטבלה Groups ושולחת את המזהה של כל קבוצה לפונקצייה getGroupFeed().
getGroupFeed(group_id) – יוצרת url לפי ה Facebook Graph API אשר שולף פוסטים של הקבוצה group_id שנוצרו ביומיים האחרונים ומכיל את השדות message, created_time, id של כל פוסט. מקבלים json עם חלק מהתוצאות, כאשר בסוף ה json מופיע next_url שמתאים לתוצאות הבאות. ב-next_url ע"מ לשלוף את התוצאות הבאות וממשיכים כך עד מתקבלות כל התוצאות. כל 'chunk' של jsonים נשלח לפונקציה getPosts(json_chunk, group_id).
getPosts(json_chunk, group_id) – הופכת את json_chunk למילון של jsonים, ושולחת כל json לפונקציה UpdatePost(post, group_id) של המודול ***Updater***.
לאחר מכן מגדיר המודול את זמן עדכון כל קבוצה להיות הזמן הכי מאוחר שפוסט בה נכתב (בעזרת שאילתת UPDATE).

Updater

מודול זה מעדכן פרטי פוסט יחיד בבסיס הנתונים. לאחר שנוצר עצם מסוג זה, ניתן לקרוא למתודת ***UpdatePost*** שלו המקבלת את הפוסט ואת ה- ***group_fb_id*** של הקבוצה ממנה הוא נשאב (ובה פורסם). בהינתן המילון של הפוסט אותו היא מקבלת כפרמטר, היא שולפת את ההודעה (אם אין – פוסט זה אינו חוקי). כעת היא קוראת ל- ***Parser*** כדי לקבל מבנה מסוג ***JobPostDetails***. לאחר מכן היא מעדכנת בטבלאות את הפרטים של הפוסט אם הוא לא מעודכן שם.

Parser

תפקידו של ה-***Parser*** הוא ניתוח טקסטואלי של הודעות מתוך פוסטים.
ה-***Parser*** מקבל תוכן של הודעה מתוך פוסט (ההודעה נמצאת בחלק ה-"message" של ה-***Json***, דוגמא בנספחים) ומעבד את המידע הטקסטואלי הנמצא בה.
הפונקציה המרכזית של ה-***Parser*** היא הפונקציה ***parse***, שמקבלת תוכן של הודעה מתוך פוסט (ההודעה נמצאת בחלק ה-"message" של ה-***Json***) ומחזירה אובייקט מסוג ***JobPostDetails***, אשר מכיל את השדות הבאים:

- *city_id* – מס' מזהה של העיר מהפוסט.
- *state_id* – מס' מזהה של המדינה מהפוסט.
- *company_id* – מס' מזהה של החברה מהפוסט.
- *emp_form id* – סוג משרה שפורסמה. 0 – משרה מלאה, 1 – משרה חלקית, 2 – משרה זמנית.
- *work_manner*– אופן העבודה. 0 – עבודה במשרד, 1 – עבודה מהבית.
- *email* – כתובת הדואר האלקטרוני בפוסט.

עוברת על המילים בהודעה ומנסה להתאים כל מילה (או צירוף של מילים) לביטוי שמופיע באחת הטבלאות הסטטיות הבאות: Companies, Cities, States. בנוסף, הפונקציה מנסה למצוא ביטויים מתאימים ע"מ לעדכן את השדות emp_form_id, work_manner, email. כאשר נמצאת התאמה, האובייקט ***JobPostDetails*** מעודכן בהתאם. יתכן שחלק מהשדות יקבלו ערך ***None*** אם ה-***Parser*** לא מצא ערכים עבורם בהודעה.

שרת ה-WEB

Server:

השרת בנוי בעזרת FLASK והוא מטפל ב-2 סוגים של בקשות:
1) 3 בקשות "סטטיות" - עיר, מדינה וחברה:
בהינתן מחרוזת השרת מחזיר רשימה של פריטים מהטבלה הרלוונטית שמתחילים באותה מחרוזת – מאפשר השלמה אוטומטית בצד הלקוח
2) בקשת חיפוש – בקשת חיפוש יכולה לכלול כל אחד מהפריטים הבאים ושילובים בינהם: מילות חיפוש, סוג משרה – מלאה, חלקית או זמנית. מדינות וערים. נכון לעכשיו החיפוש מתמקד רק בתוך ארה"ב. החיפוש בצד השרת נעשה על ידי ביצוע Join בצורה חכמה – קודם כל מאחדים טבלאות שמסננים על פיהם , על ידי Join רגיל – כדי לסנן רשומות, ורק לאחר מכן מבצעים LEFT JOIN עם שאר הטבלאות.
כמו כן השרת מחזיר תמיד את המשרות הרלוונטיות כשהן ממוינות מהחדש לישן.

Client:

הלקוח בנוי ב-Javascript בעזרת ספריית React של Facebook המאפשר רענון חכם של הדפים בהתאם למידע שהתקבל וחלוקה היררכית של האלמנטים בדף.

חבילות python בהן השתמשנו

pymysql – added this to our directory on nova

datetime

urllib

urllib2

json

sched

time

threading

unicodedata

re

sys

os

itertools

flask