# C:\Users\Awesome\Desktop\solr.png

# מסמך מערכת

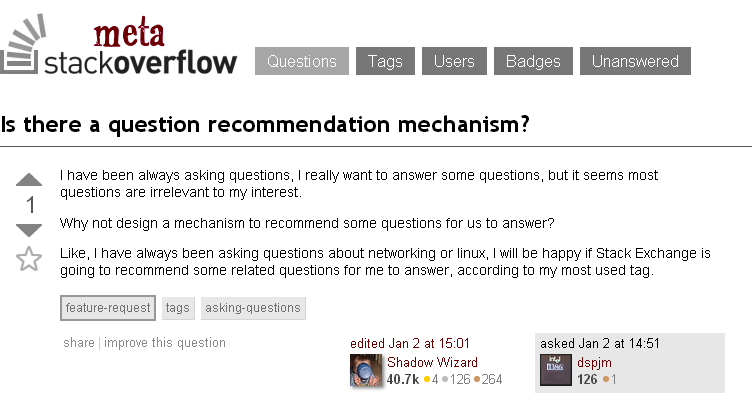
# stack**overflow** Toolkit

## כותבים :

ענבל עמרני, יזהר גלבוע, ערן הירש

# מוטיבציה

1. ניתן לראות שקיים צורך למערכת שמאפשרת למשתמש לקבל שאלות שיידע לענות עליהן.

[](http://meta.stackoverflow.com/questions/214650/is-there-a-question-recommendation-mechanism)

1. בשאלה שאין לה תשובות, מופיע בתחתית השאלה המלל הבא:



stackoverflow מעודדים משתמשים למצוא את העונה המתאים לשאלה.

1. באתר Q&A גדול אחר שנקרא Quora, יש מנגנון שמאפשר בהינתן שאלה למצוא את המשתמש המתאים לענות עליה.
2. קיים שוק חזק של אפליקציות מעל stackoverflow, האפליקציות בשוק מופיעות בקישור הבא :  
   <http://stackapps.com/questions/tagged/app>  
   המערכת שלנו תשתלב היטב בשוק הקיים.

# דרישות

הקדמה  
הפרויקט שלנו מתמקד באתר sof (stack**overflow**) כאשר נרצה לתת סט כלים על גבי האתר.  
הכלי הראשון שאנחנו רוצים ליצור הוא כלי Answerer Recommendation.

Answerer Recommendation - backend  
- בהינתן "שאלה חדשה" תחזיר את המשתמשים שהכי מתאימים לענות עליה.  
- בהינתן משתמש תחזיר את השאלות שהוא עשוי לענות עליהן.   
  
Answerer recommendation - UI

דף חיפוש  
תיבת טקסט (חיפוש) שמכניסים אליה קישור לשאלה ב- sof.  
הפלט של החיפוש יהיה משתמשים שמדורגים לפי המשתמש שהכי מתאים לענות עליה. בנוסף

למשתמש הכי מתאים ניתן להראות על איזה שאלה דומה הוא ענה.

דף רישום  
תיבת טקסט שמכניסים אליה username של משתמש ו-email.  
המשתמש יקבל למייל כל ערב רשימה של השאלות שהכי מתאימות לו לענות עליהן.

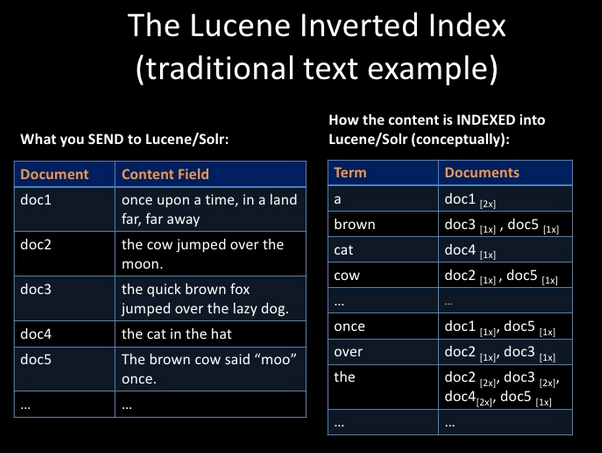
# סקירה טכנולוגית

## Solr

Solr הוא שרת חיפוש / אינדוקס המבוסס על ספריות Lucene לחיפוש ואינדוקס טקסטואלי.

## Lucene

Lucene היא ספריית חיפוש ואינדוקס טקסטואלי בעלת ביצועים גבוהים, אך ניתן להתייחס אליה כספריית Token Matching ודירוג תוצאות בעלת ביצועים גבוהים, המסוגלת לבצע חיפוש טקסטואלי. אם נתרגם זאת למונחי Machine Learning, ניתן להסיק מכך ש Lucene Index הוא בעצם מטריצה רב מימדית שעבור שדה מסוים שנאנדקס נקבל מטריצה שממפה כל Token שאונדקס לכל המסמכים המכילים אותו.  
כמו כן עבור כל מסמך ניתן לקבל את ה"וקטור" של ה Tokensהמרכיבים אותו.



# אחזור מסמכים ב SOLR\LUCENE

Lucene משלבת שתי שיטות אחזור מרכזיות שמתחלקות לשני מודלים, מודל בוליאני (BIM), שלפיו מתקבלת ההחלטה איזה מסמכים הם תוצאות החיפוש, ומודל וקטורי (VSM), שלפיו מחולק ציון לכל אחד מהמסמכים שחזרו והתוצאות ממויינות לפיו.

לפי המודל הוקטורי גם מסמכים וגם שאילתות הם וקטורים ממושקלים במרחב רב מימדי שבו כל מילה (distinct index term הוא המינוח הנכון, אך בפרויקט שלנו שבו מאנדקסים טקסט אז כל term הוא בעצם מילה באנגלית) היא מימד, והמשקל הוא הערך של המילה לפי TF-IDF. אין דרישה מיוחדת של VSM שהמשקלים יהיו ב TF-IDF, אך אלגוריתם זה לרוב מניב תוצאות איכותיות.

עוד מידע על האחזור והדירוג ב Lucene:

<http://lucene.apache.org/core/3_6_2/api/core/org/apache/lucene/search/Similarity.html>

# רכיב MoreLikeThis של SOLR

רכיב MoreLikeThis נותן את היכולת לבנות חיפוש חדש על בסיס מסמכים המאונדקסים ב SOLR על מנת למצוא מסמכים נוספים ה**דומים** להם. הדמיון מוגדר על ידי המשתמש בפרמטרים המציינים איזה שדות מגדירים דמיון, וכמה השפעה לתת לכל שדה המשתתף בהגדרת הדמיון (Boosting). ישנם פרמטרים נוספים שרצוי להגדיר, אך הם פחות חשובים וניתן למצוא את כולם בדף ה Wiki של SOLR:

<https://wiki.apache.org/solr/MoreLikeThis>

הרכיב מחלץ את ה top TF/IDF terms, בעצם את המילים הכי ערכיות במסמך, על סמך השדות שהגדרנו עבורם דמיון. לאחר מכן הרכיב בונה ממילים אלו שאילתת Solr ומדרג את המסמכים שחזרו על בסיס פונק' האיחזור הבסיסית של Lucene.

# ארכיטקטורת על

## Backend

1. את כל מאגר Stack Overflow הורדנו מהאינטרנט, כ-25GB דחוסים של טקסט.
2. הקמנו Solr על שרת וירטואלי מ-Google Compute Cloud.
3. אינדוקס 15 מיליון שאלות ותשובות של משתמשי sof מ-2008 ועד 2014 (קובץ Posts.xml).
   * בניית רכיב אינדוקס שמפרק את ה XML של הטקסט ובונה SolrDocuments, ושולח ל Solr ע"מ שמנוע ה Lucene יפרק אותם ל Tokens ויאנדקס.
4. בניית מנוע בסיסי שבהינתן מסמך (שאלה ב-sof) מוצא פריטים דומים לפי מספר מדדי דמיון :
   * השוואה של המילים בשאלות.
   * הכותרות של השאלות.
   * התגים שמצורפים לשאלות.
   * אורך השאלות.
   * טווחי הזמנים של השאלות (פחות או יותר באותם שנים).
   * כמה תשובות היו לשאלות.
5. תשתית לביצוע בדיקות אופטימיזציה  
   מציאת שאלות דומות והשוואה של המשתמשים שקיבלנו למשתמשים שענו בפועל לשאלות.

## UI

1. שימוש [בפרויקט Blacklight](http://projectblacklight.org/) כתשתית ל-UI.
2. בניית דף החיפוש למשתמש שיתקשר עם ה-Backend.
3. בניית דף רישום ומנגנון שליחת מיילים.

# איך בחרנו לאנדקס את המידע

על מנת לקשר בין משתמש לשאלה שהוא ירצה לענות עליה הזדקקנו למיפוי בין משתמש לבין העדפותיו ומומחיותו, וכן לאפשרות לשלוף על כלל התוכן הטקסטואלי של מאגר שאלות Stackoverflow במהירות ויעילות. לשם כך בנינו שני Collections ב SOLR:

1. Collection השאלות והתשובות
2. Collection המשתמשים והשאלות שעליהן בחרו לענות

חומר הגלם לשני התהליכים הוא XML ה Posts (Posts.xml). זהו קובץ XML גדול המכיל את כלל השאלות והתשובות (לא כולל תגובות, שזה עוד צורת העברת מידע שקיימת ב SOF שלא התייחסנו אליה בפרויקט) של אתר StackOverflow. נסביר את תהליך האינדוקס בכל אחד מה Collections.

# Collection השאלות והתשובות

זהו Collection פשוט המאפשר שליפה של שאלות לפי כל השדות שלהן, בעיקר על סמך שדה הטקסט, ובכך מאפשר חיפוש טקסטואלי על המאגר.

השלבים לאינדוקסו הם:

1. פירוק ה XML לשדות ובניית מסמך SolrInputDocument לפי ה API של SOLR.
2. שליחת ה XML לשרת ה SOLR לאינדוקס (תוך התחשבות בגדלי BULKים לטובת ביצועים)

בעצם בבניית ה Collection הזה לא התבצעה אלגוריתמיקה כלשהי וכל הפונקציונליות באה לידי ביטוי בקינפוג הפרמטרים של השדות. פרמטר חשוב שהגדרנו הוא TermVectors. בכל השדות שנרצה להפעיל עליהם את ה More Like This הפעלנו את האופציה של TermVectors על השדה, שזו אופציה שמאפשרת ל Solr לחשב לשדה הזה עבור כל Token (שזו מילה בשאלה או בתשובה, במקרה שלנו) את ערכי ה TF וה IDF שלו. בשלב האופטימיזציה נעשה שימוש ב TermVectors כדי לבחור מילים בצורה חכמה מתוך ה Body של השאלות כי לפעמים נצפית התנהגות לא רצויה מרכיב ה MoreLikeThis שגמישותו מוגבלת.

הקוד המאנדקס את Collection זה נמצא בנתיב:

<https://github.com/toolkitsof/sofkit/blob/master/indexer/src/Index.scala>

# Collection המשתמשים

מה שדרוש לנו על מנת להמליץ למשתמש לענות על שאלות מסוימות הוא פרופיל המשתמש והעדפותיו. אם בהינתן שאלה חדשה נוכל ליצור שאילתת חיפוש של מילות מפתח ערכיות לשאלה ולבדוק איזה משתמש הכי "קשור" למילים האלה, נוכל ככה להציע עונים פוטנציאליים לשאלות ללא מענה במאגר.

הדרך שבה רתמנו את Solr לצורך זה היא בניית מסמך עבור כל יוזר, שלכל מסמך כזה יש שדה טקסט המכיל את כל המילים של כל השאלות שהמשתמש ענה עליהן. בצורה זו, כשנשאל שאילתה שמבוססת על שאלה ללא מענה (בהמשך נסביר כיצד שאילתה זו נבנית) נדע שסט התוצאות ידורג לפי שקלול של כל המילים הרלוונטיות בין כל המשתמשים האחרים כפי שמבטיח אלגוריתם Tf-Idf לדירוג התוצאות.

בבניית Collection זה התבצעה אלגוריתמיקה כבדה יותר מאשר ב Collection הקודם. האתגר המרכזי בביצוע החישובים והאגרגציות שנדרשים לבניית ה Collection הוא כמויות המידע הגדולות שבתהליך בודד לוקח זמן רב לעבד. מסיבה זו בחרנו בתשתיות מקביליות כמו Spark, שהיא תשתית לחישוב מבוזר מבוססת שפת Scala שגם היא שפה שניתן בקלות לבצע בה חישובים מקביליים (Scala = Scalable Language). לכן את תהליך האינדוקס כולו כתבנו ב Scala ובחלקו השתמשנו בספריות Spark לצורך אגרגציה מקבילית.

השלבים בביצוע האינדוקס:

1. חילוץ קובץ של כל השאלות מתוך אוסף השאלות והתשובות מה Collection שבנינו קודם.
2. יצירת מיפוי בין כל משתמש לשאלה שהוא בחר לענות עליה על בסיס הקובץ הקודם.
3. כאן נעשה שימוש ב SPARK לצורך מקביליות פעולת הקיבוץ. Spark יודע למקבל את התהליך גם בין מעבדים שונים באותו מחשב וגם על פני מספר מחשבים במידה והוקם Cluster של Spark המורכב מיותר ממחשב יחיד.
   1. ביצוע Group By על הקובץ הקודם לפי מזהה משתמש.
   2. עבור כל משתמש, שלוף את התוכן של השאלות שלו מה Collection הקודם לפי מזהי השאלות שקובצו.
   3. בנה מסמך SolrInputDocument על בסיס התוצאות.
   4. שלח את המסמך ל Solr לאינדוקס. גם כאן בדומה לשלב הקודם מתבצע שימוש ב TermVectors על השדות הטקסטואליים.

הקוד המאנדקס את Collection זה נמצא בנתיב:

<https://github.com/toolkitsof/sofkit/blob/master/indexer/src/>

כששם הסקריפט מקביל לסעיף המדובר בשלבים.

מקורות:

<http://lucene.apache.org/core/3_6_2/api/core/org/apache/lucene/search/Similarity.html>

<https://wiki.apache.org/solr/MoreLikeThis>

<https://cwiki.apache.org/confluence/display/solr/MoreLikeThis>