

# DARE YOU GO



נכתב ע"י:

- אוויס מוסא
- מוחמד ווֹתָד

מנחה: כפיר לב-ארי

# תוכן עניינים

1 .....	מוטיבציה .....
1 .....	Dare you go .....
2 .....	הארcitקטורה של המערכת .....
2 .....	חלוקת העבודה .....
3 .....	רשימת פיצ'רים .....
3 .....	יצירת משתמשים ורישום במערכת .....
4 .....	יצירת dare ע"י משתמש .....
4 .....	הוספה dare ל- comments .....
5 .....	הוספה dare proofs .....
5 .....	.....
5 .....	הכרזה על proof כמנצח .....
6 .....	סינון dares השמורים במערכת לפי פילטרים .....
7 .....	חיפוש משתמשים .....
7 .....	עדכנים מחברים .....
7 .....	.....
7 .....	הציג תוכן עדכני לאחר swipe down .....
8 .....	עדכן אוטומטי של רשימות המידע .....
8 .....	טעינה הדרגתית של משימות .....
8 .....	רישום למשימה מסויימת .....
9 .....	פרופיל אישי משתמש .....

9 .....	
9 .....	חיפוש שימושות לפי מילוט מפתח (keywords)
10 .....	תכל
11 .....	PARSE
12 .....	Properties
15 .....	Data Access Layer
16 .....	Builder Design Pattern
17 .....	IMPLEMENTATION
26 .....	בדיקות
28 .....	מה הלהה

# מווטיבציה

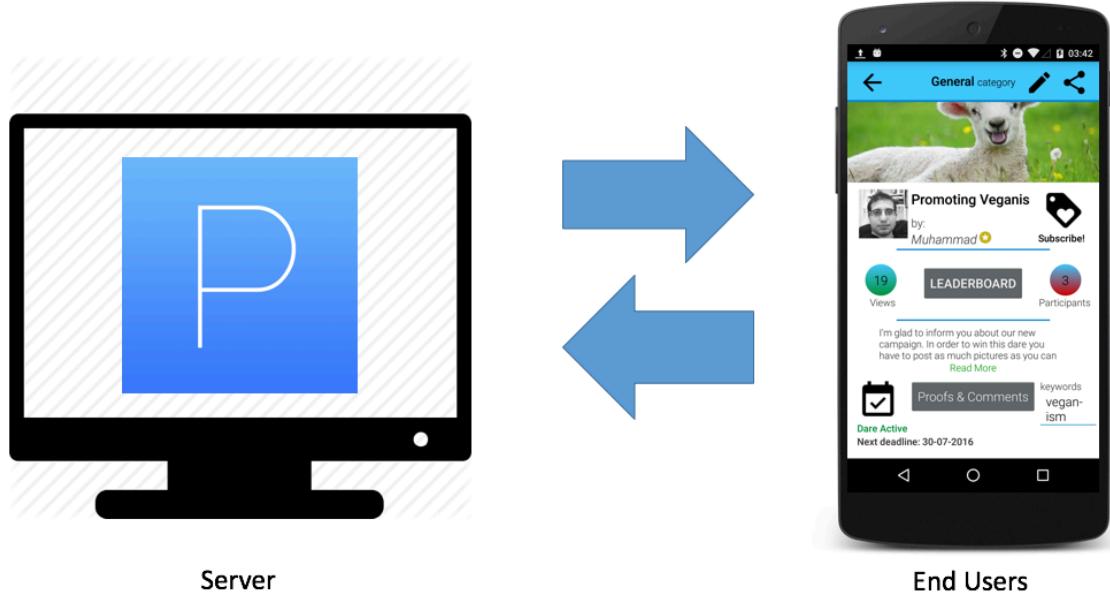
היום רוב האנשים מחוברים כל הזמן לרשומות החברתיות בכלל ולפייסבוק וטוויטר בפרט. בשנים האחרונות נפוצה תופעה חדשה ברשומות החברתיות שלא כל כך הינו רגילים לה בעבר. נראה שהאנשים מתעניינים מאוד גם משתתפים באטגרים שהחברים שלהם ברשת החברתית מעלים. דוגמא טובה לאטגר זה הוא אטגר דלי הקרכח שמטרתו לעורר מודעות למחלת ALS. יש כמובן אינסוף דוגמאות של אטגרים אחרים כמו קמפיינים לגיוס כספים, תחרויות, בקשות סייע לנזקקים, או שאלות בנושאים מגוונים שהאנשים מעלים לרשת ומחייבים לשובה מחבריהם. פייסבוק וטוויטר כיום נתנות מענה לטרינד זהה אבל לא מתחמים בו. **כאן Go You Dare**

נכנסת לתמונה!

## DARE YOU GO

Dare You Go היא אפליקציית אתגר-מענה. משתמשי האפליקציה מעלים חידון/אתגר/שאלה, והם גם יכולים לחת חלך ולענות על אתגר שימוש אחר העלה. אתגר יכול להיות סרטון וידאו, טקסט או תמונה. המשתמשים יכולים לפגג לתשובות שהעלו אחרים באמצעות ליק. האדם שמחלית מי המנצח הוא זה שייצר את המשימה. זכייה במשימות מגדילה את הניקוד של המשתמשים. משתמשים שמתעניינים במשימות המפורסמות ע"י משתמש ספציפי יכולים פשוט להוסף אותו כחבר ואז יקבלו עדכונים על כל משימה שהוא מפרסם!

## הארQUITטורה של המערכת



המערכת שלנו מורכבת משרת Parse בצד השני. העברת המידע נעשית בשני הצדדים, מהשרת אל ה- Android device והפוך. התקשרות בין שוניים עברת דרך השירות Android devices

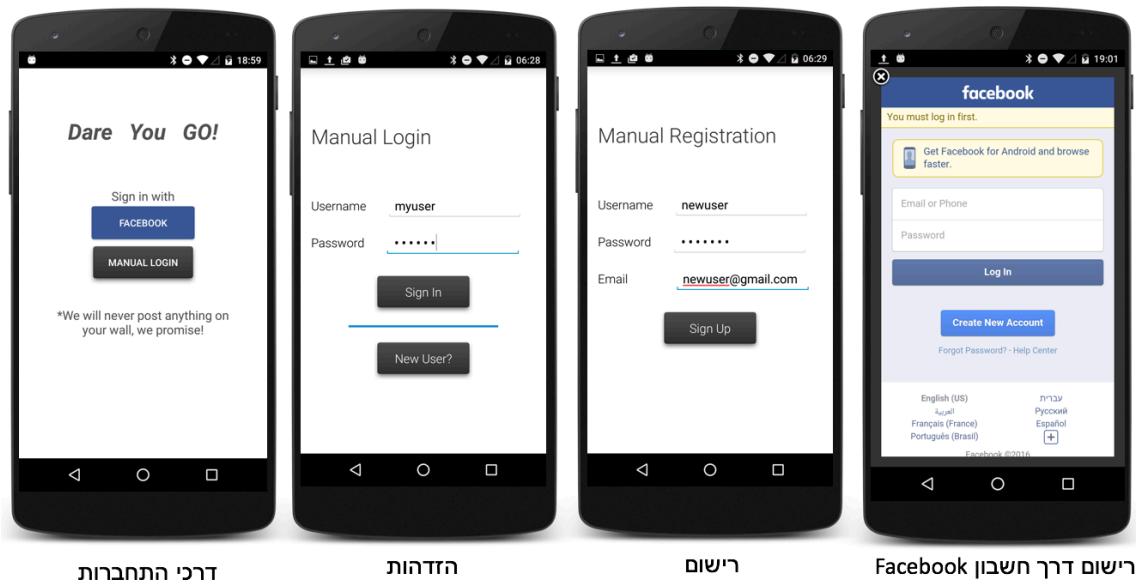
## חלוקת העבודה

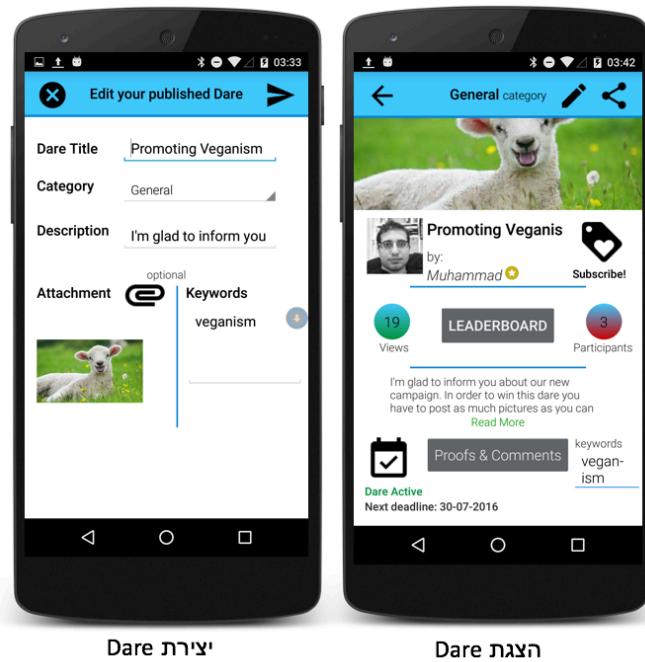
הפרויקט בוצע ע"י שתי קבוצות, הקבוצה הראשונה התמקדה בחווית השימוש ובממשק המשתמש (GUI), והקבוצה השנייה (אנחנו) התמקדה במודל של המערכת. החלק שלנו הכיל למעשה את כל הfonקציונליות של האפליקציה. כדי שנוכל לשלב את העבודה של שתי הקבוצות הוגדר ממשק בינו. לצורך מימוש הדרישות מהמערכת היינו צריכים להקים שרת משלנו שיטפל בבקשתו ויאחסן את הנתונים או לעבוד מול פלטפורמה אחרת. משום שראינו שישנן פלטפורמות קיימות אשר עוננות על כל הדרישות שלנו ואף מעבר להן בחרנו לעבוד עם פלטפורמה זאת. הפלטפורמה ששימשה אותנו נקראת PARSE (עליה נרחיב בהמשך).

## רישימת פיצ'רים

### יצירת משתמשים ורישום במערכת

בפעם הראשונה שמשתמשים באפליקציה יש צורך לבצע רישום במערכת ע"י בחירת שם משתמש, סיסמה ואימייל או להירשם בעזרת החשבון בפייסבוק. בהפעולות הבאות של האפליקציה המערכת מזיהה את המשתמש וכן אנחנו חוסכים ממנו להכנס את הנתונים שוב. כדי שנוכל להשתמש בחשבון facebook וחשבון章程 רגיל במערכת הינו צריכים לעשות קישור בין חשבון פייסבוק וחשבון Parse שנთמך ע"י הפלטפורמה.



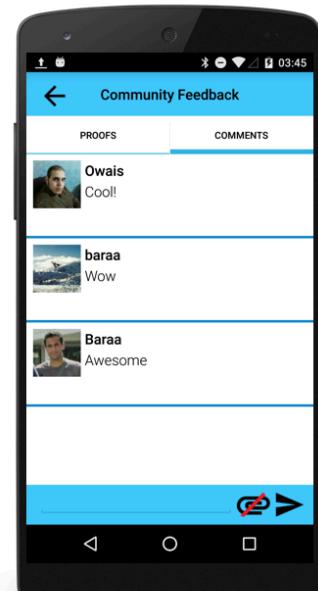


**יצירת dare ע"י משתמש**

כל משתמש המחבר למערכת (sign in) יכול ליצור משימות (dares). אלה יופיעו בתפריט המשימות באפליקציה ויאפשרו למשתמשים אחרים להשתתף בהן. משימה מוגדרת ע"י כותרת (למשל: שאלת מתמטית), קטגוריה אותה ניתן לבחור מຕוך רשימה מוגדרת מראש, תיאור של הרשימה בעזרת טקסט, deadline שהוא המועד האחרון להעלאת פתרון או הוכחה למשימה, קובץ (ויאו או תמונה) ו- keywords (שימושים לצורך חיפוש משימות במערכת).

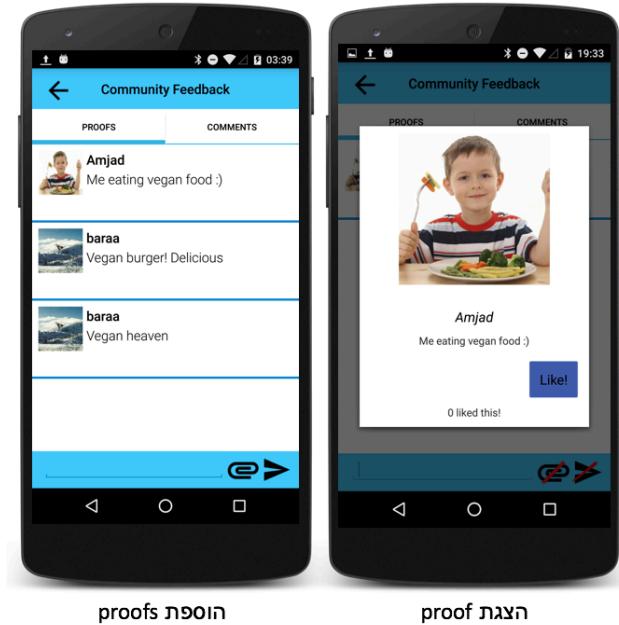
## הוספה dare ל comments

dare הוא טקסט שמගיבים באמצעותו ל- comment מסויים. כל comment משוויר מעצם הגדרתו למשימה גם למשתמש שהעלה אותו. באפליקציה אנחנו מבדילים בין שני סוגי של תגובות אפשריות לדare. הראשונה היא טקסט שמගיבים באמצעותו לדare אך לא נחשב כ-proof, השניה היא חלק מה-proof עצמו.



הוספה comment

## dare → proofs



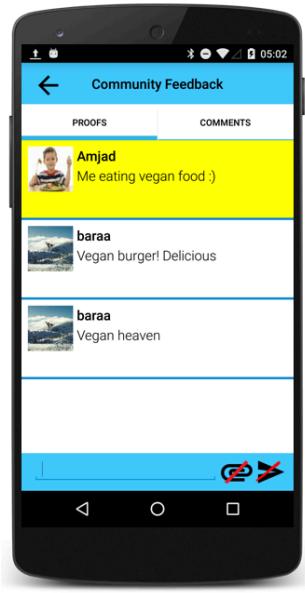
הוספה proofs

הציג proof

כדי להשתתף במשימה ספציפית המשתמש מעליה proof שאמור לייצג את הפתרון שלו למשימה או לאתגר. Proof יכול להיות טקסט, תמונה או סרטון וידאו. המנץח במשימה הוא זה שיבחר ע"י המשתמש שייצר אותה. משתמשים אחרים יכולים לעשות ליק proof של מישהו אחר.

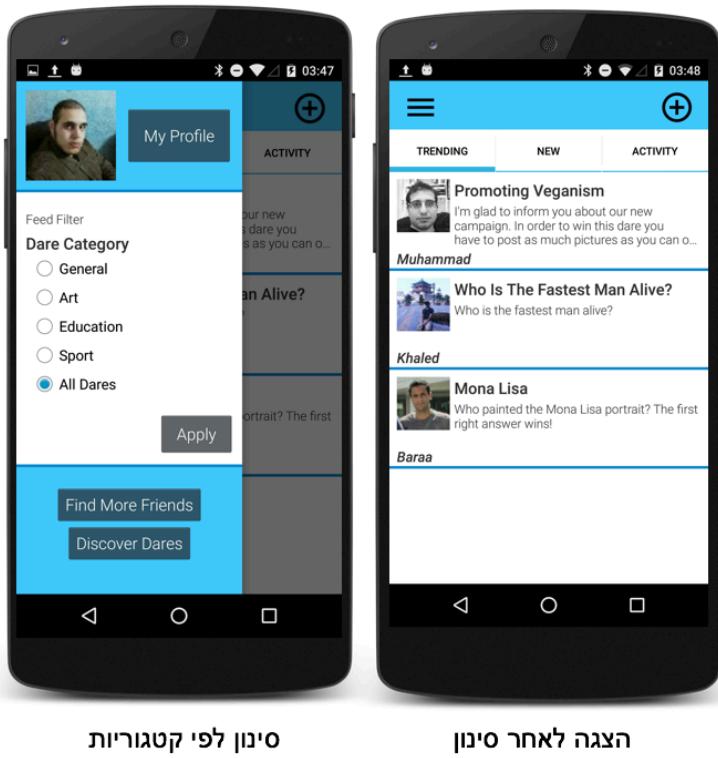
## הכרזה על proof כמנצח

המשתמש שייצר את המשימה הוא זה שבוחר את ה proof המנצח. proof מנצח מגדיל את הניקוד של המשתמש שהעלה אותו.



בחירה מנצח

## סינון *dares* השמורים במערכת לפי פילטרים



סינון לפי קטגוריות

הצגה לאחר סינון

אחת מהדרישות מהאפליקציה היא לאפשר למשתמשים לצפות בסוג מסוים של שימושות לפי העדפתם האישית ולאו דווקא לראות את כל המשימות ששמורות במערכת. כך למשל משתמש שמתעניין בספרות יוכל לצפות ארכטוגריה שלחן בספרות. בנוסף לאפשרות אלה, התפריט הראשי מכיל אינפורמציה על 3- קבוצות של שימושות: Trending, New ו- Activity, כאשר:

▪ **Trending:** לכל שימוש יש ניקוד שקבע כמה הוא *trending*. ניקוד זה מושפע מכמה גורמים שונים שמשלם לא בהכרח שווה:

- הוכחות (*proofs*) שימושים אחרים מעליים
- הערות טקסט (*comments*) שימושים אחרים מעליים
- מס' האנשים שעוקבים (*subscribed*) אחרי המשתמש

בתפריט זה המשימות מופיעות כשהן ממוננות לפי הניקוד שנקבע לכל אחת מהן בסדר יורד ובהתאם להגבלות מסוימות שיבחרו ע"י המשתמש (כמו קטgorיה מסוימת).

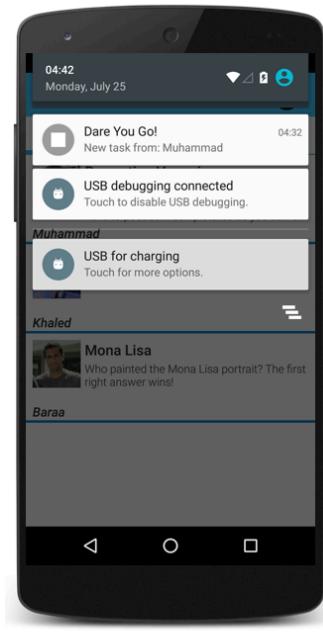
▪ **New:** לכל שימוש יש תאריך יצירה. תאריך זה משמש אותנו למילוי המשימות, כך שבוסףו של דבר המשתמש יוכל לראות את המשימות החדשנות לפני הישנות ולהיות מעודכן בכל מה חדש באפליקציה. גם כאן כמו ב- *Trending* אנחנו מאפשרים מילוי תחת הגבלות שיבחרו ע"י המשתמש ובהתאם לקטgorיה שיבחר.

▪ **Activity:** אלה משימות שהועלו ע"י המשתמש או ע"י חברים וגם משימות שהמשתמש עשה להן subscribe. בתפריט המשימות *Activity* יופיעו כל

המשימות שבמערכת שעונות על הדרישות שהוזכו וגם על הקטגוריה  
שבחר המשתמש.

## חיפוש משתמשים

באמצעות פיצ'ר זה, המשתמש יכול למצוא משתמשים אחרים הרשומים  
במערכת, להוסיף אותם כחברים (או להסיר אותם) ולצפות בפרופיל שלהם.

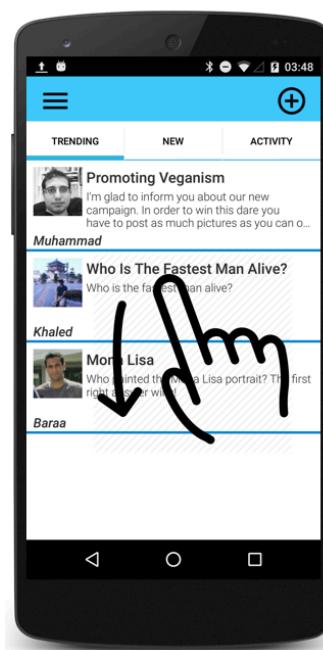


### עדכון מוחברים

משתמשים במערכת יכולים להוסיף  
משתמשים אחרים כחברים שלהם.  
אם המשתמש Bob מוסיף את Alice  
חברה שלו, אז כל פעם ש- Alice  
עלתה משימה דרך האפליקציה  
Bob יקבל נוטיפקציה שהועלתה  
משימה חדשה ע"י חברתו Alice.



חיפוש משתמשים



### הציג תוכן עדכני לאחר swipe down

כדי לחדש את הרשימות השונות באפליקציה עם המידע  
העדכני, אנחנו מאפשרים למשתמש לטען את המידע ע"י  
פעולות swipe down. פיצ'ר זה נתמך ברשימות של  
המשימות Trending, New, Activity וגם ברשימות של proofs  
וה-comments.

## עדכון אוטומטי של רשימות המידע

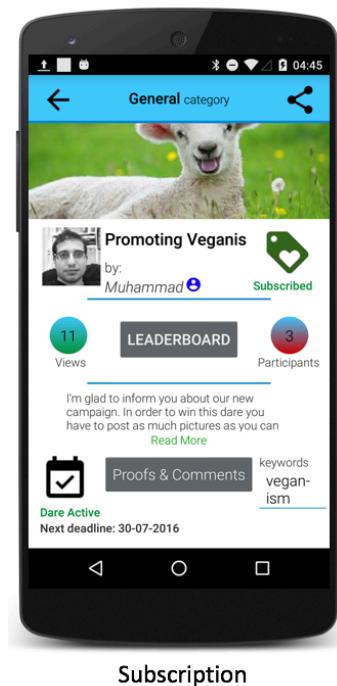
כדי שהמשתמש יתעדכן במידע העדכני בזמן שהוא משתמש באפליקציה, אנחנו דואגים שהמידע החדש יוצג מיד ובלי שהמשתמש יצטרך לבצע פעולה refresh דינית.

## טעינה הדרגתית של מושימות

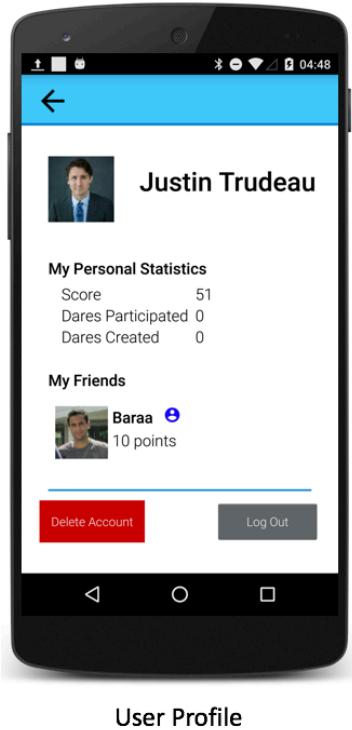
כדי לשפר את הביצועים של האפליקציה אנחנו מאפשרים טעינה של מושימות ב- batches במקום לטעון את כלון ביד. כך אנחנו חוסכים בזמן הנטנה של המשתמש ומסתפקים בטעינת המידע רק כאשר בו צורך. בנוסף לשיפור ביצועים אנחנו מורידים גם מהעומס על השרת.

## רישום למשימה מסוימת

כדי לעקוב אחר מושימות, אנחנו מאפשרים למשתמש לעשות subscribe למשימה. לאחר הרישום, המשימה תופיע בתפריט הראשי תחת Activity. ישנה גם אפשרות לבטל את הרישום במידה והמשתמש מפסיק להתחנין במשימה.



## פרופיל אישי משתמש



User Profile

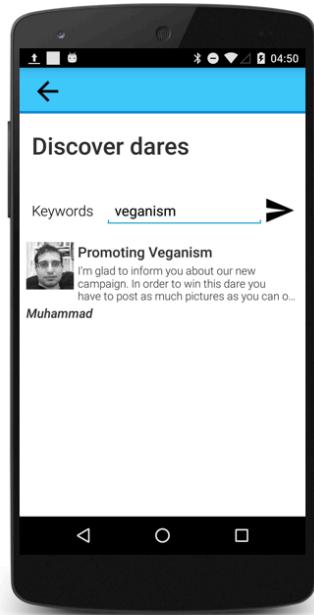
לכל משתמש רשום במערכת יש פרופיל משתמש. הprofil כולל:

- תמונה פרופיל
- שם המשתמש
- הnick שצבר
- מס' המישימות שהשתתף בהן
- מס' המישימות שהוא יצר
- רשימת חברים (רק למשתמש יש גישה לרשימת החברים שלו).

דרך הprofil המשתמש יכול למחוק את החשבון שלו ולבצע log out.

## חיפוש מישימות לפי מילות מפתח (keywords)

בעת יצירה חדשה אנחנו מאפשרים למשתמש להגדיר עבורה מילות מפתח. מילים אלה מאוד שימושיות לצורך חיפוש מישימות. מנגנון זה דומה מאד ל- hashtag ברשתות החברתיות פייסבוק וטוויטר.

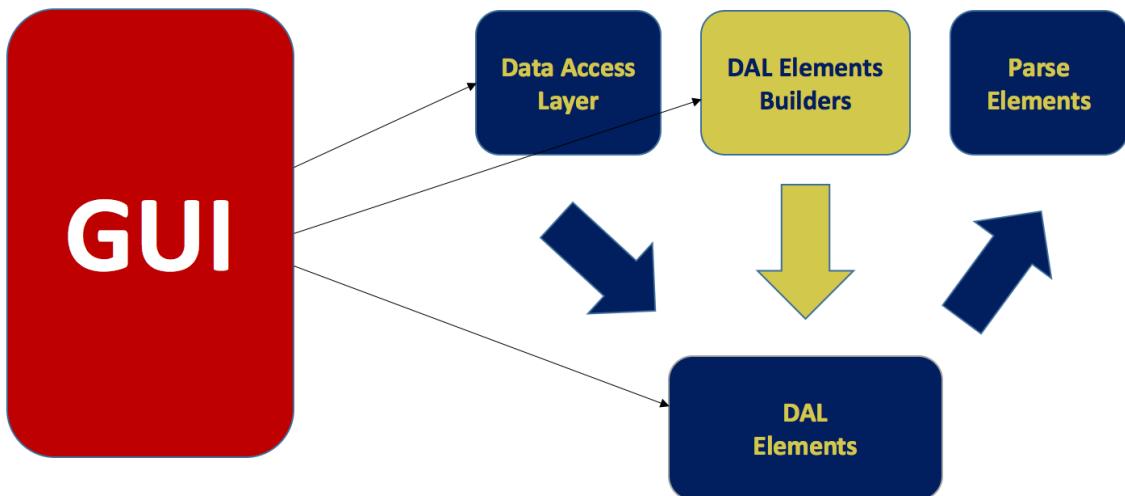


חיפוש לפי מילות מפתח

## תיכן

החלק של הקבוצה שלנו התמקד בתמיכה בכל הfonksiyoniot הנדרשת מהמערכת דרך ממשק קל שישמש את הקבוצה השנייה, תוך הקפדה על הפרדה בין העבודה של שתי הקבוצות, הפרדה בין המודול וה- GUI. לצורך כך חשפנו לפני ה- GUI שכבת ביניים דרכה הוא מקבל את הנתונים או מעדכן אותם. שכבה זו מאפשרת לצלות ה- GUI להעתלם לגמרי מחלקי הקוד שנוגעים ממש לפלטפורמה (Parse) שימושה אוננו. מבחיננתם הפלטפורמה ב- backend הייתה לגמרי שקופה. השיקול הזה היה חשוב מאד מבחיננתנו כבר מתחילה העבודה על הפרויקט, רצינו שהקוד של הקבוצה האחראית על ה- GUI יהיה תלוי תלויה בפלטפורמה שימושה אוננו ב- .backend

התרשימים הבא משקף את התוכן של המערכת:

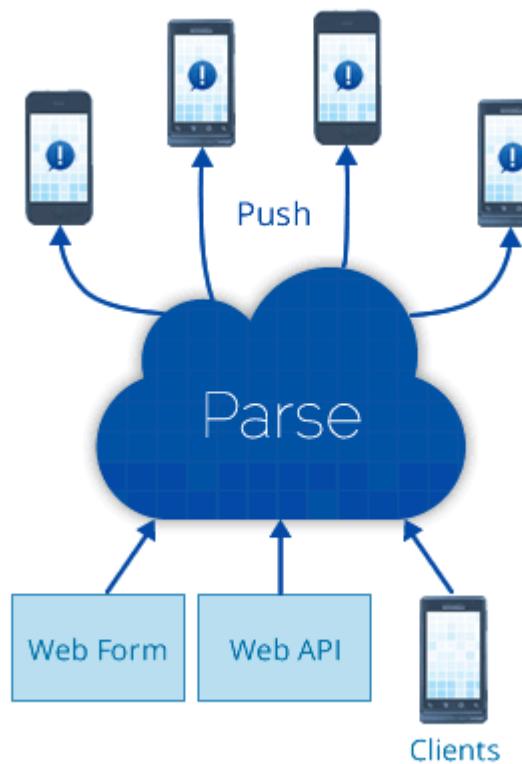


.DAL (Data Access Layer) ■

כפי שניתן לראות מהתרשימים, הביקשות של ה- GUI נשלחות לשככת הביניים כך של- GUI אין בכלל גישה ל- Parse Elements, אלמנטים של הפלטפורמה (משם). הדרך היחידה של ה- GUI לעדכן את הנתונים ב- Parse או לקרוא אותם היא דרך השכבה שהגדכנו בלבד. מבחינת ה- GUI, השכבה Data Access Layer היא דרכן single entry point למודול, כל גישה ל- Parse חייבת לעבור דרך. כך אנחנו מבודדים את ה- GUI מהשרת ומנטרלים את התלות בין השניים. ל- GUI ישנה גישה ליצירת האלמנטים שעוטפים את האלמנטים של הפלטפורמה. יצירת אלמנטים אלה נעשית לפי builder design pattern שעליו נרחב בהמשך.

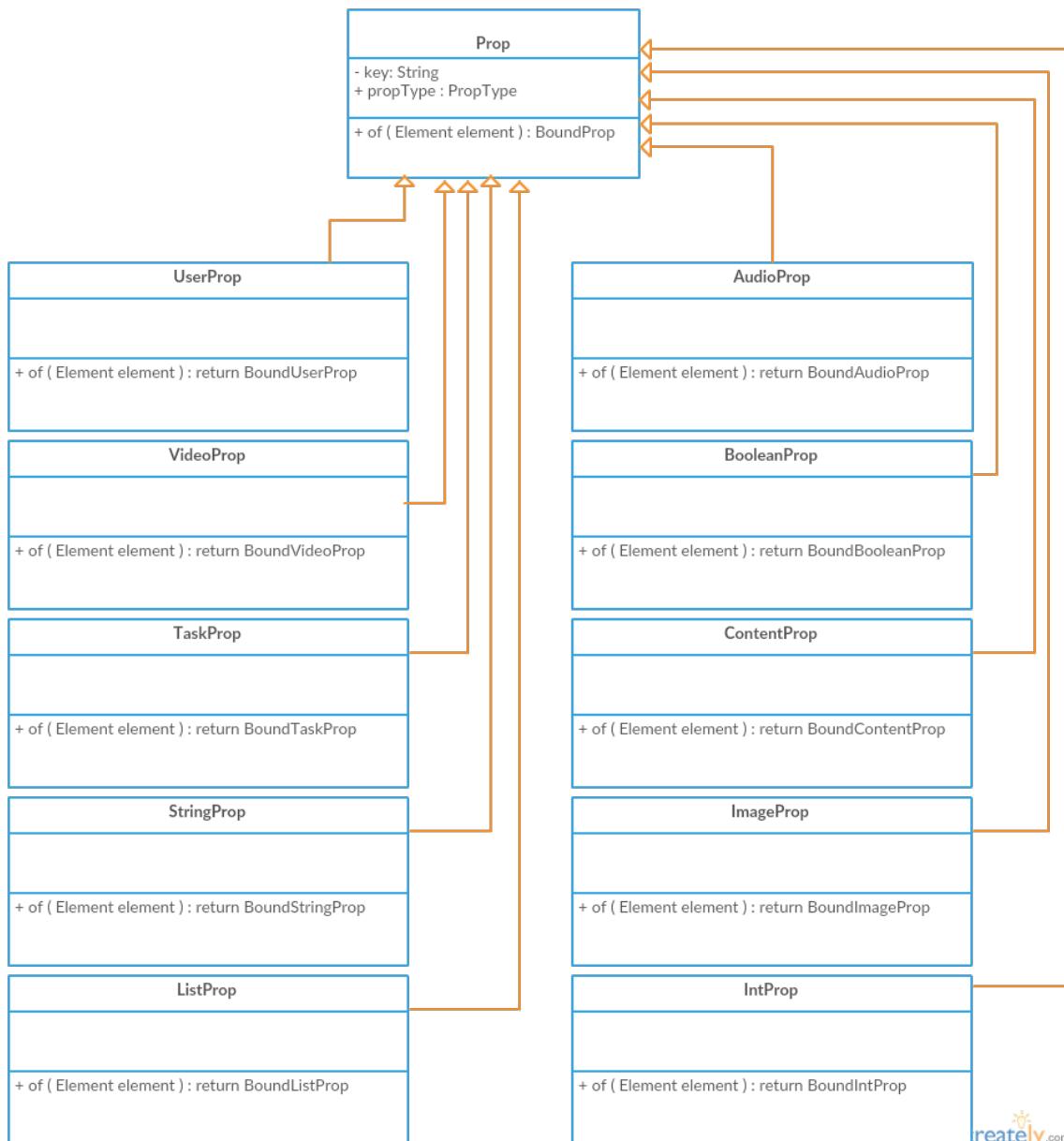
## PARSE

זו היא פלטפורמה אשר משמשת כ- backend solution עבור אפליקציות מובייל (ולא רק). הפלטפורמה תומכת בהמון פיצ'רים אוטם היינו צריים בשבייל פיתוח האפליקציה. הפלטפורמה מאפשרת שמירה של אובייקטים, ביצוע שאלות בסיסד הנתונים שבענן, שליחת נוטיפקציות, ניהול משתמשים, שמירת קבצים, ניהול מידע בצורה אסינכרונית, אינטגרציה עם פייסבוק ועוד. העבודה עם הפלטפורמה חסכה מأتנו המון כתיבת קוד והקלה באופן משמעותי בIMPLEMENTASI הדרישות מהמערכת.



## Properties

שמירת ה- `data` ב- Parse נעשית באמצעות Key-Value pairs. כאשר ה- `Key` מוצג ע"י מחרוזת וה- `Value` יכול להשתיר לтиיפוסים שונים (מספר, מחרוזת, קובץ ועוד). כדי להקל על עצמנו בימוש ולא להתעסק יותר מדי עם המחרוזות שמייצגות כל אחד מהאובייקטים שהומרו במערכת החלטנו להגדיר properties. כשאנו מגדירים property אנחנו מעבירים אליו את ה- `key` ושובחים מזה בהמשך.

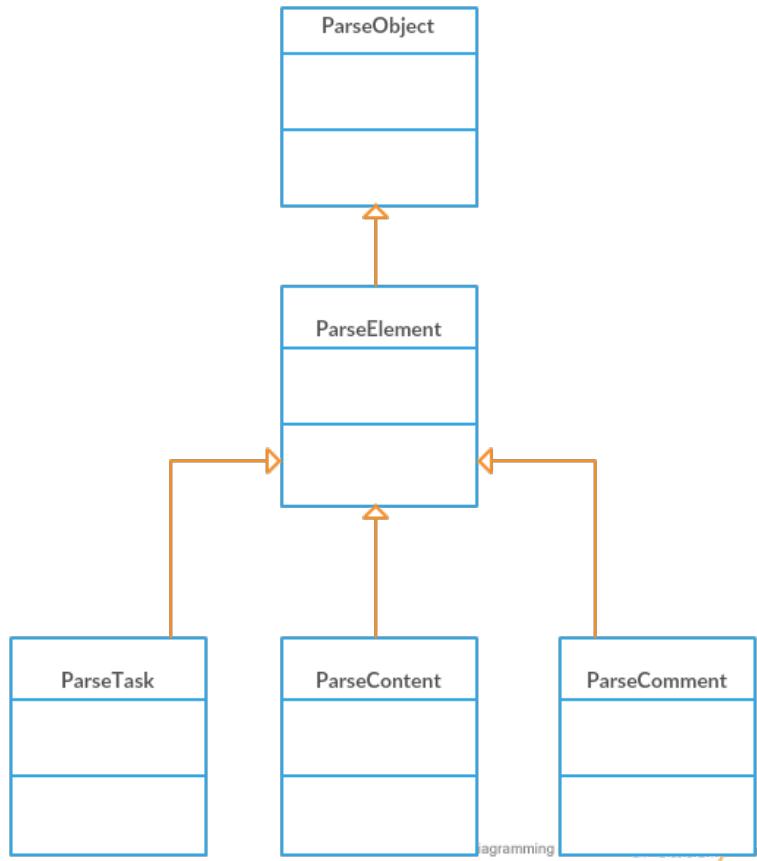


הגדרת ה- `key` כשלעצמה אינה מספקת וצריך AiCsho לקשר בין האובייקט ב- `Parse` והתמונה עצמה. לצורך כך הגדרנו כחלק מהמחלקה `Prop` את המתודה `of` שמקבלת את האובייקט כפרמטר ומחזירה אובייקט חדש `BoundProp` המכיל את כל המידעה הדרישה: גם האובייקט ב- `Parse` וגם ה- `Key`. סוגים שונים של `properties` מצריכים הגדרת `key` properties שונים. טיפוסים אלה מופיעים ב- `class` `diagram` בעמוד הקודם. כפי שניתן לראות כל ה- `properties` יורשים את המחלקה `Prop` ומעמיסים את המתודה `of`. טיפוס האובייקט המוחזר ע"י המתודה `of` בכל אחת מה- `derived classes` יורש את המחלקה `BoundProp`.

אם למשל אנחנו רוצים שלאובייקט שמייצג משימה ב- `Parse` תהיה תכונה מסוימת כמו הכוורתה של המשימה, אז אנחנו מגדירים `StringProp` ומעבירים אליו את ה- `Key` שאנו נבחר. ב- `StringProp` מוגדרת המחלקה `BoundStringProp` שמאפשרת לעדכן את ה- `Value` של ה- `Key` שבחרנו (של הכוורתה). בהמשך אם נרצה לעדכן את הכוורתה כל מה שציריך לעשות זה לקרוא למתודה `of` שיש לה גישה ל- `Key` וגם לאובייקט ב- `Parse`.

בנוסף ל- `Key` שאנו מעבירים ל- `property` בעת יצירתו אנחנו מחייבים אם הוא יהיה אופציוני או חובה. למשל אנחנו מגדירים את הכוורתה לכל משימה כחובה, וכך שניסיון יצירת משימה בלי להעביר מחרוזת כפרמטר בשבייל הכוורתה יכשל (נMARK exception). אנחנו נסביר בהמשך איך אנחנו אוכפים את החוקים האלה.

ראינו עד כה מהי תוכנה ואיך היא תורמת לנו ומקלה בפיתוח. נעברו עתה לתייאור המחלקות שגדירות את התוכנות עברו כל אחד מאובייקטי המידע שאנו חנו מתחזקים ב- Parse:

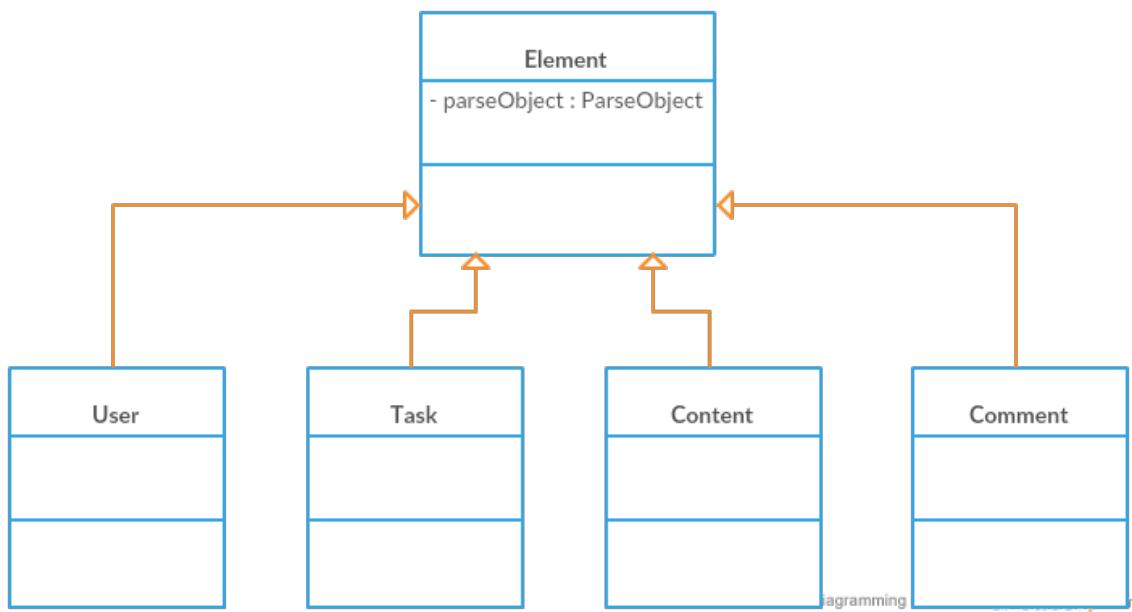


המחלקה הראשית ParseObject היא המחלקה המייצגת אובייקט ב- Parse. כפי שכבר אמרנו, המידע ב- Parse נשמר כ- Parse.key-value pairs. ParseElement מגדיר את ההתנהגות המשותפת של המחלקות ParseContent, ParseComment ו- ParseTask. ParseContent מגדיר את התוכנות שמאפיינן אובייקט המיצג Content. ParseTask מגדיר את התוכנות שמאפיינן אובייקט המיצג Task. Content ו- Task בתחוםם. התוכנות מוגדרות כשדות סטטיים קבועים כדי שהם יהיו תקפים לכל האובייקטים שייבנו בהמשך. למשל, לכל האובייקטים המיצגים Task יהיו מפתחות זרים ב- Parse ולא יקראו בשמות שונים. בחירת התוכנות של כל אחד מהאובייקטים נעשית כmbn לפי המידע שאנו רוצים לשירות לאובייקט והטיפוס שלו. ב- Task למשל אנחנו רוצים תוכנה שתציג את הכתובת של המשימה והיא צפוי מסווג StringProp. למעשה דרך המחלקות הנ"ל אנחנו קובעים

את התוכנות ובוחרים עבורן מפותחות. בשיטה זו אנחנו יכולים תמיד להרחב ולהוסיף תכונות לאובייקטים לפי הצורך ובצורה קלה מאוד.

## Data Access Layer

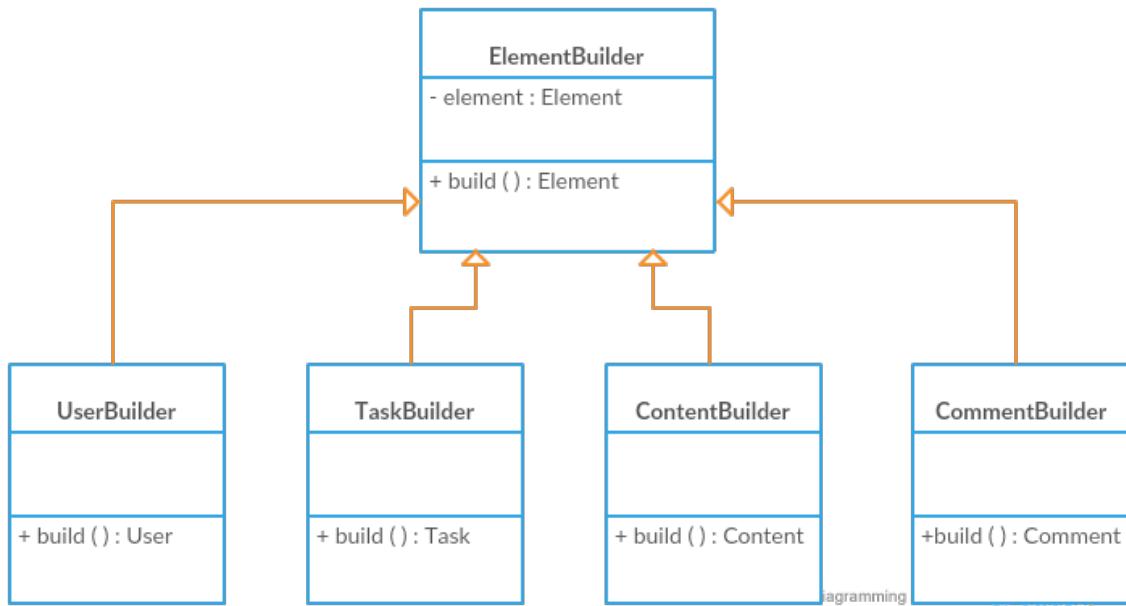
בנוסף למחלקות שתוארו קיימת היררכיה נוספת מקבילה המתארת את הפעולות שניתן לבצע על האובייקטים. המחלקות בהיררכיה זו מהוות את הממשק שאנו נוחשים כלפי חוץ ומשמש את הקבוצה השנייה האחראית על ה- GUI.



כפי שניתן לראות מה- Class Diagram אחד משדות המחלקה הוא אובייקט מטיפוס ParseObject. על מנת לקשר בין ה- derived classes והתכונות שהגדכנו קודם העברנו ב- Constructor של כל אחת מהמחלקות אובייקט מטיפוס המחלקה המקורי בהיררכיה שהציגנו קודם. למשל, ב- Constructor של Comment אנו מעתירם אובייקט מטיפוס ParseComment ושמורים אותו בשדה parseObject וכן נוצר הקשר בין אובייקט מטיפוס Comment והתכונות שהגדכנו. כדי לחיב את עצמנו ואת הקבוצה השנייה להגדיר את האובייקטים בעלי לשכו להעביר את הפרמטרים שאנו מוצאים להם ושהגדכנו אותם כתכונות, הגדרנו את ה- Constructors של המחלקות User, Task, Content, Comment ו- User כ- protected, השתמשנו ב- Builder Design Pattern לייצרת האובייקטים ודאגנו לפניה יצירה האובייקט שככל הפרמטרים הועברו.

## Builder Design Pattern

על מנת להפריד בין ייצרת האובייקטים וייצוגם בחרנו ליצירת האובייקטים את התבנית Builder Design Pattern



תבנית זו הינה מאוד שימושית במקרים רבים כי הרבה מהתכונות שאנו מגדירים עבור האובייקטים הן אופציונליות. ב- **Builder** של **Constructor** של אנו מחייבים את המשמש להעביר את הפרמטרים שהם בוגר חובה ואת האופציונליות אנחנו מאפשרים להעביר דרך מתודות נוספות. שילוח הבקשות ל- **Parse** לא מטבחת כל עוד ה- **build** לא נקרא וכך אנחנו יכולים מאוד במס' הבקשות שימושurate ל- **Parse**.

ב- **Builder** של **Constructor** אנחנו מעבירים את **element** שיכל להיות **parseObject** של **User**, **Task**, **Content**, **Comment**. נזכיר שהאחד השdots של **element** הוא **parseObject**. כדי לוודא שהמשמש העביר את כל הערכים הדרושים אנחנו מבצעים במתודה **build** בדיקה ומודאים שעבור כל תוכנה שהוגדרה כחובה הועברו ערכים ע"י המשמש. בדיקה זו אפשרית ב- **Java** בזכות ה- **Reflection**. בדיקת האימות נעשית על המשתנה **parseObject** שמוגדר כshedah במחלקה **Element**.

## IMPLEMENTATION

נתיחה עתה למחלקות הקשורות לייצוג של האובייקטים השונים:

### משתמשים:

המחלקה User:

- *path: DareYouGo/src/com/dareyougo/dal/User.java*

מחלקה זו מאגדת את כל האינפורמציה הקשורה למשתמש במערכת:

- שם המשתמש
- רשימה של החברים
- תמונה פרופיל
- ניקוד של המשתמש
- מזהה של חשבון הפיסבוק (אופציונלי)

מחלקה זו גם אחראית על כל הפעולות הקשורות למשתמש:

- עדכון ניקוד
- הוספה חבר
- הסרת חבר
- החזרת ה- *proofs* שזיכו את המשתמש
- החזרת רשימת ה- *dares* שנוצרו ע"י המשתמש
- קביעת נתונים מזהים ותמונה פרופיל

המחלקה UserBuilder:

- *path: /DareYouGo/src/com/dareyougo/dal/UserBuilder.java*

משמשת לייצירת אובייקטים מסוג *User* ואוכפת שכל ה- *data* הנחוץ לייצירת האובייקט הועבר ע"י המשתמש.

## משימות

### המחלקה ParseTask:

- path: DareYouGo/src/com/dareyougo/dal/ParseTask.java

מחלקה זו מאגדת את כל התכונות (properties) שמאפיינות משימה במערכת:

תכונות חובה:

- כוורת
- המשמש שיצר את המשימה
- תאריך היצירה
- מספר הצפויות

תכונות אופציונליות:

- תוכן
- תיאור באמצעות טקסט
- קטגוריה
- נראות (Public/friends only)
- מילوت מפתח (Keywords)
- דירוג של המשימה (משמש אותנו כדי למיין את המשימה בתפריט *Trending*)
- מועד אחרון להשתתפות במשימה (Deadline)
- מועד לאחר ה- deadline שבו יוכraz המנץח. עד למועד זה המשתמשים יכולים לעשות לייק להוכחות של משתמשים אחרים.

### המחלקה Task:

- path: /DareYouGo/src/com/dareyougo/dal/Task.java

מחלקה זו אחראית על הפעולות שנitinן לבצע על אובייקט מסווג *Task*:

- מethodot get/set לנתונים הבאים:
  - מועד אחרון להשתתפות במשימה
  - מועד לאחר ה- deadline שבו יוכraz המנץח
  - דירוג של המשימה
  - מס' הצפויות במשימה

- המשמש שיצר את המשימה
  - הcotרתת של המשימה
  - תוכן המשימה
  - תיאור המשימה
  - הקטגוריה
  - הנראות של המשימה
  - מילוט מפתח המזוהה על המשימה
- אתחול מס' הצפויות במשימה ועדכנו
  - החזרת התגובה בסדר יורד לפי מס' הלוקים או לפי סדר ההכנסה

המחלקה :TaskBuilder

- *path: /DareYouGo/src/com/dareyougo/dal/TaskBuilder.java*

מחלקה זו משמשת לייצור אובייקטים מסווג Task ואוכפת אתחול של כל ה- data הדרוש (כל תכונה שהיא חובה חייב להיות מאוחלת אחרת נערך exception.).

## **סינון `dares` השמורים במערכת לפי פילטרים**

מיון המשימות נעשה בעזרת מנגנון השאלות הנתרmr ע"י Parse ומאפשר דרך נוחה (constraints) להחזרת האובייקטים כשם ממוינים ועונים על הגבלות (constraints) מסויימות.

הקוד למיון המשימות מופיע במחלקה DataAccessLayer

- *Path: /DareYouGo/src/com/dareyougo/dal/DataAccessLayer.java*

במחלקה זו מופיעות המתודות האחראיות על המיון של המשימות:

- `getNewestTask`: להחזרת המשימות ממוינות בסדר יורד לפי תאריך היצירה.
- `getTrendingTasks`: להחזרת המשימות ממוינות בסדר יורד לפי הניקוד.
- `getFeed`: להחזרת המשימות של המשתמש ושל חבריו.

## **חיפוש משתמשים רשומים במערכת**

כדי לאפשר למשתמש לחפש משתמשים אחרים שנרשמו כבר במערכת (בדומה לפיצ'ר "Find Friends" בפייסבוק), ממשנו את המתודה `:searchUsers`

- *Path: /DareYouGo/src/com/dareyougo/dal/DataAccessLayer.java*

מתודה זו מקבלת מחרוזת ומחפשת משתמשים רשומים במערכת שהם מכיל את המחרוזת המועברת. מנגנון השאלות ב- Parse מאפשר ביצוע סינון כזה באמצעות קביעת constraints, במקרה שלנו המשתמשו ב- `WhereContains`. אם המשתמש ביצע את ה- `in`oga דרך חשבון הפיסבוק הוא יכול גם לחפש בין חבריו בפייסבוק שרשומים אצלו במערכת כמשתמשים, דבר המאפשר ע"י API.

## **comment**

### :ParseComment

- *Path:* /DareYouGo/src/com/dareyougo/dal/ParseComment.java

מחלקה זו מאגדת את כל התכונות (properties) שמאפיינות comment במערכת:

תכונות חובה:

- הטקסט
- המשמש שהעלה את ה- comment
- המשימה (dare) "শ্মাচিলা" את ה- comment
- התאריך

תכונות אופציונליות:

- תגובה מנחת - כן/לא? (רלוונטי כשה>tagובה היא חלק מ- *proof*)
- תוכן הנלווה לtagובה (רלוונטי כשה>tagובה היא חלק מ- *proof*). נתייחס בהמשך לאובייקט שמייצג תוכן במערכת.
- המשמשים שעשו לijk לtagובה (רלוונטי כשה>tagובה היא חלק מ- *proof*).

### :Comment

- *Path:* /DareYouGo/src/com/dareyougo/dal/Comment.java

מחלקה זו אחראית על הפעולות שנitin לבצע על אובייקט מסווג Comment:

- מethods *set/get* לנ נתונים
- סימון לijk או הסרתו

### :CommentBuilder

- *Path:* /DareYouGo/src/com/dareyougo/dal/CommentBuilder.java

מחלקה זו משמשת לייצרת אובייקטים מסווג Comment ואוכפת אתחול של כל ה- data הדריש.

## **content**

לפי התכן שבחרנו, proof הוא content ו- comment, כאשר content יכול להיות תמונה או סרטון וידאו. תארנו מוקדם את הייצוג של comment, נעברו עתה לייצוג של content.

### :ParseContent

- */DareYouGo/src/com/dareyougo/dal/ParseContent.java*

מחלקה זו מאגדת את כל התכונות content properties) שמאפיינות content במערכת:

- טקסט (אופציונלי)
- תמונה (אופציונלי)
- סרטון וידאו (אופציונלי)
- קובץ אודיו (אופציונלי)

### :Content

- *Path: /DareYouGo/src/com/dareyougo/dal/Content.java*

מחלקה זו מגדרה את הפעולות שניתן לבצע על אובייקט מסווג Content. הפעולות הן get/set ל-data.

### :ContentBuilder

- *Path: /DareYouGo/src/com/dareyougo/dal/ContentBuilder.java*

המחלקה משמשת ליצירת אובייקטים מסווג Content ואוכפת העברת של כל הפרמטרים הדורשים לצורך כר.

כפי שראינו מוקדם, אחת מהתכונות האופציונליות ל- comment היא content proof. ולכן כדי להוסיף proof לשימוש מסויימת, כל מה שצריך הוא ליצור content שהיה חלק מ- comment. זה מדגים את השימוש בתכונות האופציונליות שהוגדרו במחלקה ParseComment.

## המחלקה :DataAccessLayer

- Path: /DareYouGo/src/com/dareyougo/dal/DataAccessLayer.java

המחלקות שהוצגו מקודם מייצגות בעיקר מידע ולא ממשה פונקציונליות מורכבות.  
במחלקה זו נמצאת הלוגיקה היותר מורכבת של האפליקציה מלבד עדכון מידע  
ויצירת אובייקטים:

- חיפוש *dares* המכילים keyword מסוים.
- החזרת רשימת ה- *new/trending dares*.
- החזרת רשימת המשתמשים של המשתמש וחבריו (*Activity tab*)
- בדיקה אם משתמש רשום למערכת.
- החזרת רשימת המשתמשים ממוינים לפי ציון.
- חיפוש משתמשים במערכת.
- חיפוש משתמשים במערכת (*Sing in/sign up* (רегистрация))

## Notifications

באפליקציה אנחנו משתמשים על מנגנון ה- Parse push notifications שנותר ע"י Parse. באמצעות מנגנון זה אנחנו מאפשרים למשתמשים השונים במערכת להתחדר בונגע למשימות שמתפרסמות ע"י חברותם. כך שכל פעם שחבר מפרסם משימה חדשה מתקבלת נוטיפיקציה אצל המשתמש. מנגנון ה- Parse push notifications ב- parse עובד לפי channels. כאשר משתמש יוצר משימה הוא מבצע את ה- push על העורך האישי שלו והחברים שלו שמאזינים על העורך מתעדכנים אוטומטית. בנוסף לעדכונים האלה אנחנו דואגים שהמידע המוצג למשתמש יתעדכן בצורה אוטומטית כשהנעים שינויים ע"י משתמשים אחרים. כך למשל אם המשתמש משתמש מסתכל על שימוש המשימות החדשות וממשתמש אחר מעלה משימה חדשה, אנחנו רוצים שהמשתמש יתעדכן מיד. באפליקציה אנחנו אומנםאפשר טעינה ידנית ע"י המשתמש (ע"י פעולה swipe down), אלא שאנו רצינו שגם במקרה ברגע שיש לנו מידע חדש ב- backend נוכל לעדכן את ה- GUI בלי פעולה יזומה של המשתמש. כדי לעשות את זה הינו צריכים גם אוסף האובייקטים (למשל המשימות) המוצגות למשתמש יתעדכו לפי מידע חדש שנקבע ע"י המשתמשים.

### מחלקה DalContainer

- Path: /DareYouGo/src/com/dareyougo/dal/DalContainer.java

מחלקה זו מייצגת רשות של אובייקטים מטיפוס גנרי (משימות, הערות,...). דרך מחלקה זו אנחנו מנהלים את הטענה ההדרגתית של הרשומות. מכיוון שככל המידע שאנו מתחזקים במערכת שומר ב- Parse אנחנו מעבירים לבניי של המחלקה אובייקט שמייצג שאלתה. דרך אובייקט זה אנחנו מחליטים מה האוסף יכול (משימות חדשות, משימות trending,...). אחד השדות החשובים במחלקה זו הוא `listener` מטיפוס `DataContatinerOnChangeListener`. התפקיד של `listener` זה הינו חשוב מאוד על מנת לעדכן את ה- GUI כשהמידע ב- backend מתעדכן. אוסף המידע עצמו מחייב לעורצים הרלוונטיים. כשישנו עדכון במידע האוסף מתעדכן ע"י עצמו וקורא ל- `notification` `listener` שנקבע ע"י ה- GUI. כדי לא לתקוע את ה- GUI בזמן טעינת המידע החדש מ- Parse השתמשנו ב- `ListenableFuture` שמאפשר את טעינת המידע על חוט נפרד והרצה של קוד ה- `listener` שנקבע ב- `frontend`. כשהטעינה מסתיימת.

## המחלקה Notification

במחלקה זו אנחנו מתחזקים מיפוי בין הערכאים והמידע הרשם לעדכנים מאוטם ערכאים. דרך מחלקה זו אנחנו קובעים אילו DalContainers רשומים לעדכנים מאילו ערכאים, ואת כדי שנעדכן את המידע ברגע שנרשם עדכון דרך הערכץ. המתודות המרכזיות במחלקה זו הן `onReceive`, `subscribe` ו- `push`:

- `subscriber`: מקבלת `subscriber` ושם הערכץ ורשותת את ה- `Subscribe` לערכץ.
- `onReceive`: מקבלת שם של ערכץ וקוראת למетодה `receiveNotification` של כל אחד מה- `subscribers` שמקשייבים על אותו ערכץ כדי לעדכן את המידע.
- `Push`: מקבל אובייקט ושם הערכץ ומבצעת `push` דרך `Parse`.

## FFmpeg

Parse – Parse היא כפלטפורמה מגבילה את גודל הקבצים שאפשר להעלות ל- 10 megabytes. באפליקציה אנחנו מאפשרים לשימוש להעלות חלק מה- proof. Parse סרטוני וידאו. היינו לכן צריכים למצוא פתרון עקיף לביעית הגבלת הנפח ב- Parse. הביעה נפתחה בעזרת דחיסת הקבצים. הרכבנו את איקות הסרטון אבל הרווחנו את היכולת לשמר את התוכן ב- parse. לדחיסת קבצי הוידאו השתמשנו בפלטפורמה ffmpeg. פלטפורמה זו מיועדת לטיפול בקבצים בכלל, וקבצי וידאו וודיו בפרט. פלטפורמה זו מאפשרת בין היתר: filtering, streaming, decode, encode.

## בדיקות

לצורך בדיקת תקינות המערכת כתבנו טסטים שבודקים תרחישים שונים. הטסטים עוזרו לנו לעלות על באגים שהכנסנו בטעות במהלך הפיתוח. רצינו לבדוק שבקוד שכתבנו אין באגים ולכן היינו צריכים לבדוק אותו מהפלטפורמה שימושה אותו בפיתוח (Parse). לצורך כך בחרנו לעבוד עם mock objects שمدמים את העבודה מול Parse אולם זה לא הצליח מכיוון ש- Parse חיבר סביבת Android.

- בדיקות אלה מופיינות בפרויקט נפרד DareYouGoTest.

בשלב השני ניסינו לעקוב את הבעה ע"י בדיקה בעזרת robolectric – פלטפורמה שמשכתבת את כל המחלקות של SDK Android ומאפשרת להריץ את הקוד על MVN רגיל. מצערנו גם ניסיון זה לא צלח.

בסוף דבר החלפנו לכתוב בדיקות אינטגרציה שרצות מול Parse במקום בדיקות היחידה. טסטים אלה בדקו בצורה מקיפה הרבה הרצה תרחישים ע"י הריצה של האפליקציה על emulator ובדיקת מקרים שונים. החיסרונו של טסטים כאלה הוא בתלותם בגורמים שאינם בשליטתנו כגון ה- device-network.

- בדיקות אלה מופיינות בפרויקט נפרד DareYouGoIntegTest.

## סביבה עבודה

- ניהול גרסאות: לצורך עבודה נוחה על הקוד ומעקב אחר השינויים של שתי הקבוצות ושל כל אחד מאתנו, בחרנו לעבוד עם *Git* (מערכת לניהול גרסאות מבוצרת) והמשק הגרפי *SourceTree*. ניהול ה- *issues* (באגים העיקריים) נעשה דרך *GitHub*.

<https://github.com/kfirlevari/DareYouGo> :repository ▪

- סביבת פיתוח: *Eclipse*

## מה הלאה

- ✓ באמצע העבודה על הפרויקט הודיעה פיסבוק על הפסקת השירות של Parse ב- 28.1.2017. שימושות הדבר היא שהאפליקציה לא תהיה יותר שימושית אלא אם עוברים לפלטפורמה אחרת בהתאם לפרסום של החברה. קבוצה אחרת שתיה מעוניינת להמשיך לפתח את הפרויקט תצטרכן לקחת זאת בחשבון.
- ✓ האפליקציה כפי שהיא היום תומכת בכל הפיצ'רים שהוגדרו בתחילת העבודה על הפרויקט. במהלך העבודה עלו כמה רעונות לפיצ'רים שיכולים לשפר את האפליקציה עוד יותר אך לא מומשו:
  - סינון של המשימות לפי קרבה למפרטים. פיצ'ר זה הינו מאוד שימושי מכיוון שהמשתמשים נוטים להתעניין במה שקרה בהם בסביבתם הרבה יותר מאשר מה שקרה במקומות מרוחקים בעולם.
  - הרחבה האינטגרציה של האפליקציה ברשותות החברתיות, כמו למשל שיתוף משימות בפייסבוק וטוויטר.
  - פיתוח *web & iOS applications*.