בס"ד

ChatApp

מסמך תיאור הפרויקט

מגישים:

מיכאל שחר

עידו בן-אל

מיכל בר-אילן

בצלאל מושקוביץ

תוכן עניינים:

הצגת הפרויקט:

בנינו שרת על פי ממשק המאפשר תקשורת עם שרתים נוספים המממשים ממשק זה, וכך נוצרה רשת של שרתים המחוברים אחד לשני.

הלקוחות יכולים להיות מחוברים לשרתים שונים (כל פעם לשרת אחד בלבד) - וכך יוכלו לתקשר אחד עם השני דרך רשת השרתים.
כאשר לקוח מתחבר לשרת מסויים - כל שאר השרתים המחוברים אל השרת הזה מיודעים שאותו לקוח מחובר לשם. לכל שרת יש רשימת שרתים שהוא מכיר ישירות - כלומר הוא מחובר אליהם ישירות ויש שרתים שהוא מכיר - אבל לא מחובר אליהם ישירות - כלומר הוא יכול להגיע אליהם דרך שרת אחר. כל שרת מחזיק ערוצים.
ערוץ זה קבוצה של משתמש אחד או יותר הרשומים לערוץ ויקבלו את כל ההודעות הנשלחות לערוץ זה. כאשר יש רק משתמש אחד בערוץ מסויים - ההודעות נשלחות לשרת ומשם לשום מקום. כאשר יש שני אנשים בערוץ מסויים - אזי ההודעות נשלחות במסלול בין השרתים המחברים ביניהם. כאשר יש יותר אנשים בערוץ ההודעה תישלח לכל השרתים הרלוונטים ודרכם לאותם אנשים.

הפרויקט מורכב מ-4 חלקים:

1. שרת AppEngine – אליו יהיו מחוברים לקוחות או שרתים אחרים.
2. לקוח Android – אפליקציה צ'אט הקוראת למתודות השרת הרלוונטיות.
3. mapReduce – מנגנון לסיווג ביקורות .
4. LoadBalancing– מנגנון לניתוב העומס בין השרתים.

שרת **AppEngine**:

הדרישה העיקרית מהשרת היא לאפשר תקשורת בין שרתים שונים ולכן השרת שלנו מממש interface אחיד.

ע"מ לקיים את הפונקציונאליות הנדרשת מימשנו פונקציות כך שעבור אלו שלא מקבלות פרמטרים יעבדו בGET ואלו שכן מקבלות פרמטרים יעבוד בPOST.

כדי לממש את הפונקציונאליות ולהחזיק את הנתונים אצלינו יצרנו DataStore עם מספר טבלאות שונות. כל טבלה מכילה מספר עמודות מינימלי ע"מ למנוע מצב של תלויות רבות ומידע כפול או שגוי. להלן מבנה הDataStore.

Message (sendMessage):

channel_id	user_id	text	longtitude	latitude	date_time	wasRead

Channel (addChannel):

channel_id	name	icon	is_my

ChannelToRemove (changeChannels):

channel_id	link_to_server

Server (register):

link	is_connected

User (login):

nick_name

UserOnChannel:

channel_id	user_id

UserOnChannel זו ישות מיוחדת שהגדרנו שאומרת בעצם עבור כל ערוץ אלו משתמשים רשומים אליו. אם לדוגמא בערוץ עם מזהה 1 יש 3 משתמשים, אז יהיו 3 ישויות מסוג UserOnChannel כאשר בכל אחד מהן הchannel_id יהיה 1 והuser_id יהיה הכינוי של המשתמש, כל אחד בתורו.

כעת, לאחר שאנו יודעים איך נראה הdatastore שלנו נאפיין את הפונקציות:

<u>פונקציית getServers:</u>

ניגש לישויות מסוג Server נוציא את רשימת השרתים שהם שלנו. נדע שהם שלנו ע"י עמודת is_connected. בנוסף נאחסן ישויות חדשות מסוג Server - שרתים שאנו מכירים אותם באופן עקיף, דרך השרתים שמחוברים אלינו. לבסוף נחזיר את כל השרתים המוכרים לנו (באופן ישיר או עקיף).

<u>פונקציית sendMessage:</u>

יוצרת אובייקט Message שמכיל מזהה ערוץ, טקסט, מיקום ושעת שליחה, ומאחסנת אותו ב data store. שולחת update לשרתים אחרים שהתבצעה שליחת הודעה חדשה. מחזירה 1 במקרה של הצלחה. אחרת, 0 עם הודעה שמפרטת מה השגיאה שהתרחשה.

<u>פונקציית getUpdates:</u>

מחזיר רשימה של כל ההודעות שלא נקראו מכל הערוצים שהמשתמש הנוכחי חבר בהם. במידה והחלטנו להעביר את המשתמש הנוכחי לשרת אחר אנו מציינים זאת בתוך הפלט תחת change_server. אנו עושים זאת ע"י שימוש בישויות מסוג Messsage, Channel וChannelToRemove.

<u>פונקציית getChannels:</u>

נשלוף את כל הישויות שקיימות מסוג Channel ונחזיר אותן.

פונקציית joinChannel:

מקבלת כארגומנט מזהה ערוץ, שולפת את מזהה המשתמש מגוגל, מקשרת בDS את המשתמש לערוץ, שולחת update לשרתים אחרים שהמשתמש הצטרף לערוץ. ומחזירה 1 במקרה של הצלחה. אחרת, 0 עם הודעה שמפרטת מה השגיאה שהתרחשה. אחרת, 0 עם הודעה שמפרטת מה השגיאה. נבדקים מקרים בעייתים בגינם ההצטרפות נכשלת, למשל אם הערוץ לא קיים.

פונקציית addChannel:

נוסיף ערוץ חדש כישות Channel . לפני ההכנסה נבדוק שהוא אינו קיים כבר בטבלה. בנוסף נעדכן את השרתים האחרים על הערוץ שנוסף. שולחת update לשרתים אחרים שהתבצעה הוספת ערוץ חדש. מחזירה 1 במקרה של הצלחה. אחרת, 0 עם הודעה שמפרטת מה השגיאה שהתרחשה.

פונקציית login:

בפעם הראשונה מבצעים זיהוי של גוגל, ונוסיף ישות מסוג User בה נשמור את ה-nick_name של המשתמש. בנוסף מסמנים את כל הערוצים שהוא חבר בהם (אנו יודעים זאת ע"י הטבלה UserOnChannel) כשלנו אם הם לא מסומנים כבר ככאלה (בגין משתמשים אחרים שרשומים אלינו ונמצאים בערוצים האלו). שולחת update לשרתים אחרים שהתבצעה התחברות חדשה. אלו למחיקת משתמש מערוץ מחזירה 1 במקרה של הצלחה. אחרת, 0 עם הודעה שמפרטת מה השגיאה שהתרחשה.

פונקציית logoff:

נמחק את המשתמש - ע"י מחיקת הישות המתאימה מסוג User ואם הוא המשתמש היחיד שעבורו אנו מחזיק ערוצ/ים אז נסמן את הערוצ/ים הללו כשלא שלנו (is_my = False) מחזירה 1 במקרה של הצלחה. שולחת update לשרתים אחרים שהתבצעה התנתקות חדשה. אחרת, 0 עם הודעה שמפרטת מה השגיאה שהתרחשה.

פונקציית changeChannels:

מאחסנים את הרשימה של מזהי הערוצים ולידם את שם השרת שאליו יש להעביר את המשתמשים בתור ישויות מסוג .ChannelToRemove ואז getUpdates תשלוף את הישויות האלו ותבקש מכל אחד מהמשתמשים בערוצים אלה לעבור לשרת הנ"ל. אנחנו רוצים לבצע את המנגנון של loadBalancing, אז נעביר בעצמנו את המשתמשים שצריכים להיות מועברים לשרת אחר - נבצע logoff ואז Login לשרת המתאים.

מחזירה 1 במקרה של הצלחה.

פונקציית update:

בכל אחת מהאופציות אם אין מעגל אנחנו מעבירים את ההודעות הלאה לשרתים שמחוברים אלינו.

שליחת הודעה - אחסון ההודעה אצלנו ב dataStore כישות sendMessage

יצירת ערוץ - אחסון הערוץ החדש (אם הוא אכן חדש) אצלנו ב dataStore כישות Channel

הצטרפות לערוץ - להוסיף את המשתמש למשתמשים של הערוץ ע"י יצירת ישות UserOnChannel

עזיבת ערוץ - למחוק את המשתמש מרשימת המשתמשים של הערוץ - מחיקת הישות המתאימה מסוג UserOnChannel ואם הוא אחרון אז מוחקים את הערוץ - מחיקת הישות המתאימה מסוג Channel

מחיקת ערוץ - רק אם מי שמחק הוא היחידי בערוץ נמחק את הערוץ - מחיקת הישות המתאימה מסוג Channel

בעצם השיטה הזו עושה מה שהשיטות האחרות עושות רק שלא צריך לבדוק את ההודעה כי ידוע שהיא תקינה וישר מבצעים אותה (מבצעים שינויים ב.(Data Store

מחזירה 1 במקרה של הצלחה. אחרת, 0 עם הודעה שמפרטת מה השגיאה שהתרחשה.

פונקציית register:

מקבלת קישור לשרת ומוסיפה אותו לרשימת השרתים המוכרים באופן ישיר ונרשמת אליו בחזרה. מחזירה 1 במקרה של הצלחה. אחרת, 0 עם הודעה שמפרטת מה השגיאה שהתרחשה.

פונקציית unRegister:

מקבלת כארגומנט את לינק שרת היעד, בודקת בDS שהשרת קיים, אם קיים מוחקת אותו מהDataStore וגם קוראת לunregister שלו. מחזירה 1 במקרה של הצלחה. אחרת, 0 עם הודעה שמפרטת מה השגיאה שהתרחשה.

פונקציית leaveChannel:

שולפת כארגומנט מזהה ערוץ, שולפת מגוגל את מזהה המשתמש, שולחת update עם ארגומנטים אלו למחיקת משתמש מערוץ ומחזירה 1 במקרה של הצלחה. אחרת, 0 עם

הודעה שמפרטת מה השגיאה שהתרחשה. נבדקים מקרים בעייתים בגינם העזיבה נכשלת, למשל אם המשתמש לא נמצא בערוץ שממנו הוא רוצה להתנתק.

פונקציית getNetwork:

ניגש לטבלת userOnChannel , נוציא משם עבור כל channel_id את רשימת user_id.
נבנה גייסון מכל הנתונים וסה"כ נקבל רשימה של ערוצים ושל חברי הערוץ שהשרת מכיר.

פונקציית getNumOfClients:

נשלוף מכל הישויות מסוג UserOnChannel את כל המשתמשים שרשומים לערוץ הנתון כפרמטר ונבדוק כמה מהם נמצאים על השרת שלנו.

פונקציית getMyChannels:

שולפת את לינק השרת, מחזירה את כל הערוצים ששייכים לשרת הנ"ל לפי הDS בפורמט JSON.

לקוח **Android**:

האפליקציה מאפשרת למשתמש לראות את המיקום של חבריו ולשלוח להם הודעות במגוון ערוצים שונים (בדומה לשליחת הודעות בקבוצות באפליקציית WhatsApp).

כמו כן המשתמש יכול להוסיף ערוץ, להצטרף לערוץ קיים ולהתנתק ממנו.

בנוסף ישנו ערוץ מיוחד שמאפשר סיווג הודעות שהתוכן שלהם הוא ביקורת על סרטים כך שמקבלים תשובה האם הביקורת חיובית או שלילית.

מאחורי הקלעים מתבצעות קריאות לשרתי AppEngine של הרשת על מנת לקבל בכל רגע את המידע המעודכן ביותר ולעדכן את שאר המשתמשים בפעולות של המשתמש הנוכחי.

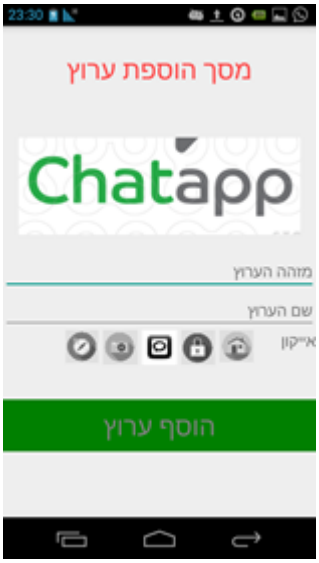
תיאור המחלקות:

AddChannelActivity.java:

מסך הוספת ערוץ.

מציג שדה להזנת שם הערוץ, מזהה הערוץ ובחירת אייקון לערוץ.

לאחר שהמשתמש מאשר את ההוספה נשלחת בקשת post לשרת.



AddChannelCall.java:

שולח את בקשת הוספת הערוץ (רגיל או ביקורת) לשרת.

BaseActivity.java:

מחלקה שממנה יורשים כל המסכים האחרים על מנת להבטיח שלא יהיו תפריטי ברירת המחדל של אנדרואיד (ActionBar).

ChannelAdapter.java:

מאפשר המרה של מידע על ערוץ לרכיב תצוגה המראה את שם הערוץ והאייקון שלו ברשימת כל הערוצים במסך ההגדרות (מזהה ערוץ לא מוצג כיוון שלא אמור לעניין משתמש מזהה של ערוץ).

בלחיצה על ערוץ מסוים במסך הגדרות תוצג למשתמש הודעה האם הוא רוצה להצטרף לערוץ זה, אם הוא בוחר להצטרף תשלח לשרת הבקשה המתאימה.

ChannelObject.java:

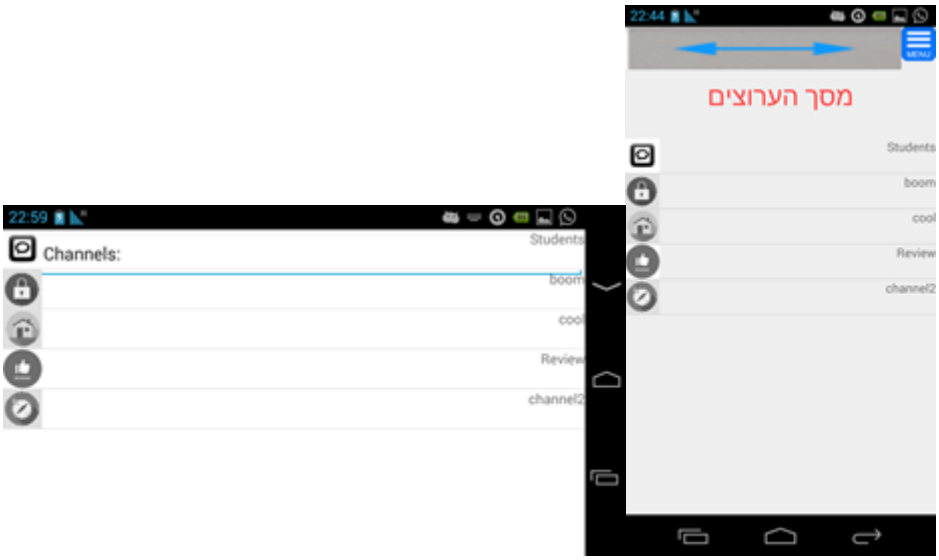
מחלקה שמייצגת ערוץ.

ChannelsActivity.java:

המסך הראשי של האפליקציה. לחיצה על כפתור חזור מתפקדת במסך זה כמו לחיצה על כפתור הבית.

מסך שמציג את הערוצים שאליהם מחובר המשתמש.

בלחיצה על ערוץ המשתמש יועבר אל מסך הצ'אט של הערוץ הנ"ל.



ChannelsFragment.java:

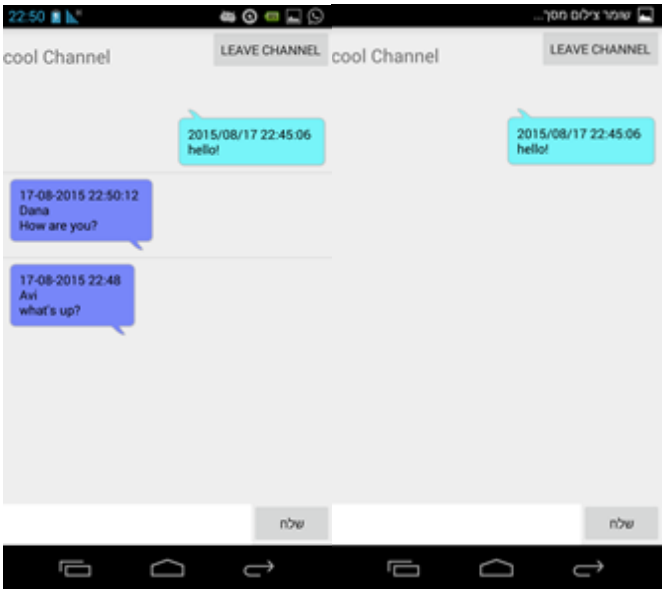
מציג את רשימת הערוצים בתוך פרגמנט במסך המפה במצב מאוזן.

ChatActivity.java:

מסך הצ'אט.

מאפשר למשתמש לשלוח הודעות לכל המשתמשים שרשומים לערוץ.

כמו כן ישנו כפתור התנתקות שמוחק את המשתמש מהערוץ על ידי שליחת בקשה מתאימה לשרת.



ChatArrayAdapter.java:

מאפשר המרה של הודעה לרכיב תצוגה מתאים להצגה במסך הצ'אט.

ChatMessage.java:

מחלקה שמייצגת הודעה.

GetCookie.java:

שליפת מזהה המשתמש של גוגל הנשמר במכשיר, כחלק מתהליך האימות שלו בכניסה לאפליקציה.

GoogleLoginActivity.java:

מסך הזדהות המופיע בהפעלת האפליקציה לאחר מסך הפתיחה.

מאפשר למשתמש להירשם אלינו לאפליקציה באמצעות חשבון הגוגל שלו הנדרש לצורך תקשורת מול השרת.

:GPSTracker.java

שולף את מיקום המשתמש.

:HttpClientHolder.java

מכיל את את האובייקט מסוג HttpClient שמכיל את הפרטים על המשתמש, לאחר שאותחל במסך הGoogleLoginActivity ע"י הפעלת מנגנון האימות שמתחיל במחלקה OnTokenAcquired, ממשיך במחלקה GetCookie ומסתיים במחלקה LoginOrLogoffCall שמבצעת קריאה למתודת השרת login (במקרה הזה, במקרה אחר לפונקצית logoff).

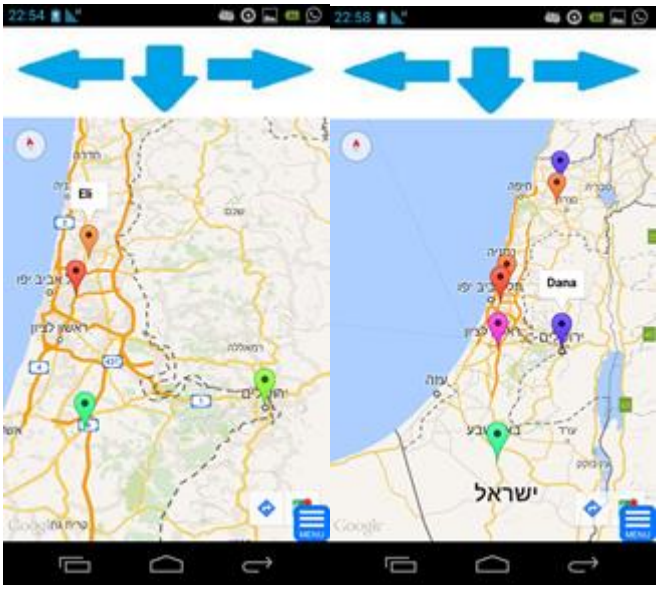
:LoginOrLogoffCall.java

שולח בקשת התחברות או התנתקות מהאפליקציה לשרת.

:MapActivity.java

מסך מפה.

מציג את המיקום של כל המשתמשים שקיימים בערוצים שאליהם אני מנוי.



:MyChannelAdapter.java

מאפשר המרה של מידע על ערוץ לרכיב תצוגה המראה את שם הערוץ והאייקון שלו ברשימת הערוצים במסך ההערוצים שלי. (מזהה ערוץ לא מוצג כיוון שלא אמור לעניין משתמש מזהה של ערוץ).

:MyGestureDetector.java

מזהה ביצוע SWIPE של המשתמש.

:OnTokenAcquired.java

מחלקה שנקראת כחלק מתהליך האימות של המשתמש.

:ReloadService.java

מחלקת עזר בשביל חץ עדכון שמופיע ע"י גלילה למטה או שקשוק המכשיר.

:ReviewChatActivity.java

מסך ערוץ ביקורת.

מממש לקוח TCP שמתקשר מול שרת TCP שמחזיר סיווג לכל הודעת ביקורת שנשלחת אליו.

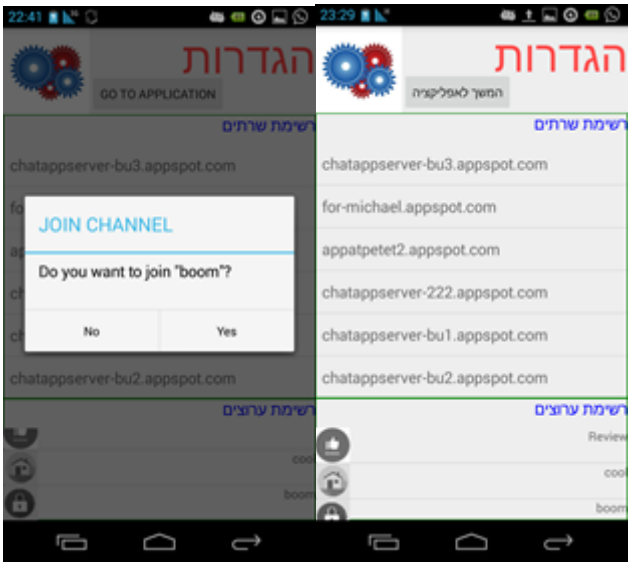
:SettingsActivity.java

מסך הגדרות.

מציג רשימה של כל השרתים המחוברים לשרת הנוכחי שמשרת את האפליקציה וכל הערוצים הקיימים ברשת.

לחיצה על שרת מאפשרת להתנתק ממנו ומעבירה למסך לוגין בכדי להתחבר אליו.

לחיצה על ערוץ מאפשרת להצטרף אליו.



<u>SplashActivity.java:</u>

מסך פתיחה.

מציג את הלוגו של האפליקציה עם טקסט מתחלף המקדים בברכה את המשתמש.

StringUtils.java:

מחלקת עזר שעוזרת לנו לזהות האם ביקורות שנכתבו בערוץ ביקורת נכתבו בשפה האנגלית.

UserHolder.java:

מאחסן את הכינוי של המשתמש (מה שלפני ה@ בכתובת המייל) ועוזר לנו לזהות את הערוצים של המשתמש.

LoadBalanceService.java:

אחראית על קריאה למנגנון הloadbalancing.

MenuAdapter.java:

מאפשר יצירת תפריט על מסך הערוצים.

MenuItem:

אוביקט המכיל מידע על פריט בתפריט שעל מסך הערוצים.

MenuFragment.java:

הפרגמנט שבו יוצג התפריט במסך הערוצים.

MapReduce:

האפליקציה מאפשרת יצירת ערוץ מיוחד שבו ניתן להכניס כקלט ביקורות של סרטים (כל ביקורת צריכה להיות באורך של 100 מילים לפחות ובאנגלית) ולקבל כפלט סיווג חיובי או שלילי עבור כל אחת מהן.

כל ביקורת תשלח דרך לקוח TCP שקיים באפליקציה אל שרת עלTCP מחשב שמריץ לינוקס ושמחובר לאותה רשת .LAN

השרת יעזר בMAPREDUCE כדי ליצור מסווג (בעת עליית השרת) על ידי ניתוח של סט אימון הכולל יותר מ1000 ביקורות מסווגות של סרטים.

לאחר יצירת המסווג השרת יאזין להודעות ביקורת מהלקוח וישתמש במסווג ובתהליך חדש של MAPREDUCE על מנת לסווג את ההודעות ולשלוח את תוצאת הסיווג חזרה ללקוח.

שרת TCP:

Server.java

בעת עליית השרת נקראת הפו' של יצירת המסווג בMAPREDUCE מתוך מחלקת Driver על ידי סט האימון.

לאחר שהמסווג נוצר אנו יוצרים לולאה אינסופית שממתינה לקבלת הודעות בsocket המוגדר מראש (IP + PORT).

עבור כל הודעה יוצרים תהליך חדש שעושה את הדברים הבאים:

*קולטים מזהה (USER + PASSWORD) מהמשתמש כדי שנדע שהמשתמש תקין.

*שולחים אישור למשתמש.

*קולטים את ההודעה עצמה מהמשתמש ורושמים אותה לתוך קובץ בתיקיה עם שמות ייחודיים על פי מונה שסופר את מספר ההודעה.

*קוראים לפו' הסיווג בMAPREDUCE מתוך מחלקת Driver כדי לדעת האם ההודעה חיובית או שלילית.

*שולחים את תוצאת הסיווג חזרה למשתמש.

Driver.java

המחלקה מכילה 2 פו':

1.יצירת מסווג:

*נעשה שימוש ב1JOB (על ידי שימוש ב: 1Mapper1 + Reducer) שבמהלכו:

**ב1Mapper אנו קוראים את כל הקבצים בסט האימון ועבור כל קובץ אנו שומרים עבור כל מילה tuple של המילה כמפתח וסיפרה (1 אם הקובץ הוא של ביקורת חיובית, -1 אם הביקורת שלילית) כערך. כאשר אנו יודעים את מאפייני הביקורת לפי שם הקובץ.

**ב1Reducer אנו סוכמים עבור כל מילה את הספרות בכל המופעים שלה ומקבלים שוב tuple של המילה וספרה כלשהי כך שהספרה מייצגת את הנטייה של המילה להופיע יותר בביקורות חיוביות (ואז מילה זו תקבל ספרה חיובית) או שליליות (ואז מילה זו תקבל ספרה שלילית).

לבסוף נוצר קובץ של tupleים שאיתו נוכל לסווג כל הודעת ביקורת בהתאם למילים שיש בה.

2.סיווג הודעה:

*נעשה שימוש בJOB2 (על ידי שימוש ב: 1Mapper1 + Reducer) שבמהלכו:

**ב1Mapper אנו קוראים את הקובץ שמכיל את ההודעה ואנו שומרים עבור כל מילה tuple של המילה כמפתח והסיפרה 1 כערך.

**ב1Reducer אנו סוכמים עבור כל מילה את האחדות עבור כל המופעים שלה ומקבלים שוב tuple של המילה וספרה כלשהי כך שהספרה מייצגת את מספר המופעים של המילה הזו בהודעה.

לבסוף נוצר קובץ של tupleים שמייצג את מספר המופעים של כל מילה בהודעה.

*נעשה שימוש בJOB3 (על ידי שימוש ב: 2Mapper2_1 + Mapper2_2 + Reducer) שבמהלכו:

**ב1_2Mapper אנו שומרים את כל הtupleים של המסווג עם פרפיקס מיוחד.

**ב2_2Mapper אנו שומרים את כל הtupleים של ההודעה עם פרפיקס מיוחד אחר.

**ב2Reducer אנו שומרים 2 רשימות: של המילים מהמסווג ושל המילים מההודעה.

עבור כל מילה מההודעה אנו מכפילים את מספר המופעים שלה בספרה המייצגת של אותה המילה במסווג, בודקים אם קיבלנו תוצאה הגיונית (למשל למילות קישור יהיו ערכים חריגים כי הן מופיעות המון פעמים ואין לנו בהכרח למידה ממספר המופעים שלהן על סיווג ההודעה) ואם כן אנו מוסיפים את התוצאה למונה.

אם בסוף התהליך המונה מכיל ערך חיובי אז נסווג את ההודעה כחיובית אחרת נסווג אותה כהודעה שלילית. הסיווג ישמר בקובץ הפלט.

*נקרא את קובץ הפלט, נמחק את התיקיות הזמניות ונחזיר את תוצאת הסיווג.

<u>לקוח TCP:</u>

<u>ReviewChatActivity.java</u>

בחלק של האפליקציה.

LoadBalancing:

מטרת מנגנון ה load balancing היא לשמור על חלוקה הגיונית ויעילה של עומסי השרתים.
מכיוון שאנו מחזיקים שרת מתוך רשת של שרתים, כשחלקם מחזיקים באותם הערוצים, נעדיף שמשתמשים מאותו הערוץ יהיו מחוברים לאותו השרת.

על מנת לנהל את העומס יצרנו מתודה בשרת אשר עוברת על רשימת הערוצים שבשרת שלנו, ועבור כל ערוץ בודקת (על פי ה id הספציפי שלו) האם הוא קיים גם בשרתים אחרים המחוברים אלינו.
במידה ומצאנו ערוץ שקיים גם אצלנו וגם אצל שרת אחר, נבדוק האם יש טעם להעביר את משתמשי הערוץ המחוברים אלינו אל השרת השני (על ידי בדיקות של מספר משתמשי הערוץ בכל שרת והיחס בן המספרים - בעזרת המתודה.(getNumOfClients -
אם מספר הערוצים שנדרשים להעברה גדול מחצי מספר הערוצים הכללי, מתודת ה Load balancing תקרא למתודות change channels שבעזרתה נשלח הודעה למשתמשי הערוץ לאיזה שרת עליהם לעבור.
בפועל, על מנת להריץ את המתודה דרך האפליקציה, יצרנו Async Task במסך המפה שיופעל ב thread נפרד עם כל כניסה למסך.
מכיוון שמימוש ה load balancer וערך החזרה אינו חלק מה interface של השרת, השרת היחיד שהאפליקציה תפעיל עליו את המתודה יהיה השרת שלנו.