זיהוי מכשיר סמארטפון בקרבת הרספברי. איך זה יעבוד?

# אופן הפעולה

* הזיהוי יתבצע באמצעות תשתית WIFI
* ה Raspberry והאפליקציה יחליפו בניהן הודעות דרך הענן, שבסיומן ידע הרספברי מי הם המכשירים הנמצאים בסביבתו, וכל מכשיר טלפון ידע מי הרספברי הקרוב אליו.
* נאמר על מכשיר שהוא *קרוב* לרספברי אם הוא נמצא באותה רשת LAN (ללא תלות במרחק הפיזי)

לפרוטוקול של זיהוי המכשירים בסביבה נקרא בשם **פרוטוקול MEP** (MAC Exchange Protocol). והוא יעבוד באופן הבא:

1. התקנת הרספברי תהיה כרוכה ברישומו בענן – כלומר בסיס הנתונים יכיל טבלה של כל מכשירי הרספברי שבשרתנו יחד עם כתובת הIP שלהם וכתובת ה-MAC שלהם.
   * *הערה*: יש לנו מכשיר רספברי אחד ולכן תהיה רק רשומה אחת בטבלה, אבל אנחנו נבנה את המערכת כך שתתאים גם לכמות גדולה של מכשירים.
2. כאשר משתמש מפעיל את האפליקציה, היא מיד תבקש מבסיס הנתונים את כתובות הIP של כל מכשירי הרספברי שתואמים את הsubnet שבו נמצא המכשיר, ואז והיא תשלח **הודעת MEP** לכל הכתובות ברשימה (הסבר נוסף בהמשך)
3. ע"ס הודעות הMEP הרספברי יוכל לרשום לעצמו בזיכרון את כתובות הMAC של המכשירים בסביבה
4. לאחר מכן הרספרי ישלח תשובה על כל הודעה שקיבל (במידת הצורך).
5. לאחר החלפת ההודעות, הן הטלפון והן הרספברי ידעו את כתובות הIP וה-MAC אחד של השני, וכמו כן הרספברי ידע בדיוק מי נמצא בסביבתו.

# מהי הודעת MEP?

* **MEP** - **M**AC **E**xchange **P**rotocol - זהו פרוטוקול החלפת כתובות MAC שאנחנו נפתח.
* הפרוטוקול יהיה מבוסס על JSON
* מבנה ההודעה:
* הסבר:
  + כל הודעת MEP מתחילה במחרוזת **$MEP**
  + **Dev Type** – סוג המכשיר. יכול לקבל רק שני ערכים: **RBPI** או **APP** (האפליקציה תשלח הודעות מסוג APP והרספברי ישלח הודעות מסוג RBPI)
  + **MAC Addr** – כתובת ה-MAC של המכשיר בצורת מספר הקסדצימלי של 12 ספרות וללא שום תווים נוספים!
  + **IP Addr** - כתובת ה-IP של המכשיר בצורת מספר הקסדצימלי של 8 ספרות וללא שום תווים נוספים!
  + **Addressee** - כתובת ה-MAC של הנמען. ניתן להעביר את הערך 0 על מנת לשלוח הודעת Broadcast לכל המכשירים הרשומים.
  + **Callback** – פעולה שאנחנו מבקשים מהמכשיר המקבל את ההודעה לבצע מיד לאחר קריאת ההודעה. הפעולות האפשריות הן:
    - **RES** (קיצור של Response) – מבקש לשלוח בתגובה הודעת MEP לשולח
    - **ACK** (קיצור של Acknowledge) – מבקש לשלוח חיווי האם ההודעה הגיעה ליעד, ללא ציון כתובת MAC ו-IP (אפשר להעביר כתובת 0 בשניהם)
    - **NOP** (קיצור של No Operation) – אל תשלח שום הודעה בתגובה.
  + **Date** – תאריך ושעת שליחת ההודעה (עדיין לא לגמרי סגור על הפורמט)
* דוגמה:

{“$MEP”:

{

“DevType”:”RBPI”,

“MACAddr”:”123456789ABC”,

”IPAddr”:”C0A80101”,

”Callback”:”RES”,

“Addressee”:”123456789ABC”,

“Date”: “28/07/2017 19:02”

}

}

# התחברות של מספר משתמשים במקביל

אם יש כמה משתמשים המחוברים לאפליקציה באותו זמן ובאותה רשת LAN אז הרספברי צריך לדעת מי מהם נשקל.

* המשתמש יצטרך ללחוץ על כפתור "התחל שקילה" רגע לפני עליה על המשקל.
* אם מישהו נשקל, והמערכת אינה מצליחה לזהות מי (למשל אם מישהו "שכח" ללחוץ על הכפתור) אז המערכת תסתכל על השקילות ב-X זמן האחרון ותנסה לנחש. זו הזדמנות טובה להתמיע אלגוריתם clustering כלשהו.
* במקרה הנ"ל האפליקציה תשאל את המשתמש "Are you Ramy Lerner?". אם המשתמש עונה בשלילה, אז התוצאה לא תישלח לענן ולא תוצג למשתמש.

# אבטחת הפרוטוקול

יש כל מיני בעיות אבטחה בשיטה הזאת לאיתור מכשירים.

אם ישאר זמן נטפל בהם כאשר הפרוייקט יהיה גמור. כרגע פחות קריטי.

חיבור באמצעות קוד QR

# אופן הפעולה

* על המשקל תימצא מדבקה ועליה קוד QR.
* הקוד הוא יחודי לכל משקל.
* לכל מכשיר רספברי יקבע מספר סידורי שיקודד לתוך הqr
* לכל אפליקציה גם יקבע מספר סידורי (צריך לחשוב איך)
* המספרים הסידוריים יהיו רשומים ב-DB (רספברי שאינו רשום בענן, ירשם לפני השימוש הראשוני בו)
* דרך האפליקציה ניתן יהיה לסרוק את הקוד ולהתחיל בשקילה.

# תקשורת בין האפליקציה לרספברי

מבחינת המשתמש, מבחינתו הוא סורק את QR ואז מיד יכול לעלות על המשקל ולראות את תוצאת השקילה באפליקציה.

ע"מ לספק את השירות הנ"ל האפליקציה והרספברי צריכים לתקשר בניהם.

* האפליקציה צריכה לדווח לרספברי שהקוד שלו נסרק (ולמסור את שם המשתמש שסרק אותו)
* לאחר מכן הרספברי צריך להתחיל לשלוח מידע ולשלוח אותו חזרה לאפליקציה.

# הודעות MEP וקודי QR

* ניזכר ברעיון מאחורי המנגנון של הודעות MEP: השתמשנו במנגנון זה בכדי שבכל רגע נתון תדע כל אפלקציה מי הם המשקלים שנמצאים בסביבתה (באותה רשת LAN), ולהיפך.
* נתקלנו בבעיה בשיטה הזאת שלפיה לא ניתן לדעת מי המשתמשים בסביבה הוא זה שנמצא על המשקל.
  + שימוש ב-QR קוד פותר את הבעיה הזאת, משום שהמרחק הפיזי בין המכשירים כבר לא רלוונטי
  + **שימוש בQR *אינו* חוסך את הצורך בתכנון של פרוטוקול להעברת הודעות** (שזה טוב לנו, כי זה אומר שהפרוייקט לא יהיה 'פשוט' מידי לצורך הציון), ז"א שאפשר להחליף את MEP בפרוטוקול יציב יותר.
* אין צורך להתעסק עם כתובות MAC ו-IP. את אלו יחליף הצורך להעביר את שם המשתמש המחובר לאפליקציה, המספרים הסידוריים של המכשירים ואת תוצאות השקילות.

# פרוטוקול DRP (Data Request Protocol)

את פרוטוקול MEP נחליף בפרוטוקול דומה בשם DRP. התשתית שלו זהה (מועבר כמחרוזת המתארת מבנה JSON ונשלח דרך azure מהרספברי לאפליקציה ולהיפך)

השדות של הפרוטוקול הם:

* **Protocol** – שם הפרוטוקול. תמיד יכיל את אותה מחרוזת "DRP$" (כדי להבדיל אותו מהודעות שאינן רלוונטיות שעלולות "להתעופף" בענן)
* **Dev Type** – אותו דבר כמו בהודעות MEP. יכיל את אחת המחרוזות APP או RBPI.
* **RBID** – המספר הסידורי של הרספברי
* **APPID** – המספר הסידורי של האפליקציה
* Username – שם המשתמש שרוצה להישקל
* **Data** – מערך אשר יכיל את כל המידע שמוחזר מהרספברי (משקל, אחוזי שומן וכו')
* **Token** – מנגנון אבטחה שימנע מאנשים לשלוח הודעות בשם משתמשים אחרים. בשלב זה נעביר בשדה זה תמיד את מחרוזת "0". נממש את זה לקראת סיום הפרוייקט.
* **Status** – שדה זה מכיל את סוג ההודעה. המכשיר המקבל את ההודעה צריך לדעת כיצד להתייחס לשדות של הפרוטוקול בהתאם לסוג ההודעה. סוגי ההודעה האפשריים הם:
  + **scanned** – מיד לאחר סריקת הקוד האפליקציה לרספברי הודעה עם הסטטוס הזה ("הרגע סרקתי אותך"). ההודעה תכיל את שם המשתמש המחובר לאפליקציה.
  + **final** – מכיל את המשקל הסופי (נשלח מהרספברי לאפליקציה). לאחר קבלת הודעה מסוג final האפליקציה סוגרת את החיבור מול הרספברי (מפסיקה לחכות להודעות נכנסות ממנו).
  + **data** – מעביר מידע לאפליקציה (בשדה data) בלי לסגור את החיבור.
  + **Exception** – ההודעה מכילה בשדה data הודעת שגיאה או אזהרה כלשהי.
  + **ack** – בכל פעם שהרספברי/אפליקציה מקבל הודעה (עם סטטוס שונה מ-ACK) הוא יענה בהודעת ACK. בהודעת ACK חשוב שימצאו המספרים הסידוריים (שאר השדות יכולים להכיל זבל).
* **Date** – תאריך ושעת שליחת ההודעה (עדיין לא לגמרי סגור על הפורמט)