|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | TAULOGOblue%201 | |  |
| אוניברסיטת תל-אביב | | **Tel-Aviv University** |
| הפקולטה להנדסה  בי"ס להנדסת חשמל | |  | | Faculty of Engineering  School of Electrical Engineering |
| מערכת לניהול ולהרצת אנליזות בתחום חקר המוח | | | | |
| פרויקט מס' 15-1-1-774  דו"ח סיכום | | | | |
| מבצעים: | | | | |
|  | שלי בר | | XXXXXXXXX | |
|  | רז רגב | | XXXXXXXXX | |
| מנחים: | | | | |
|  | אורי אוסמי | | אוניברסיטת ת"א | |
|  |  | |  | |
| מקום ביצוע הפרויקט:  בית הספר סגול למדעי המוח | | | | |

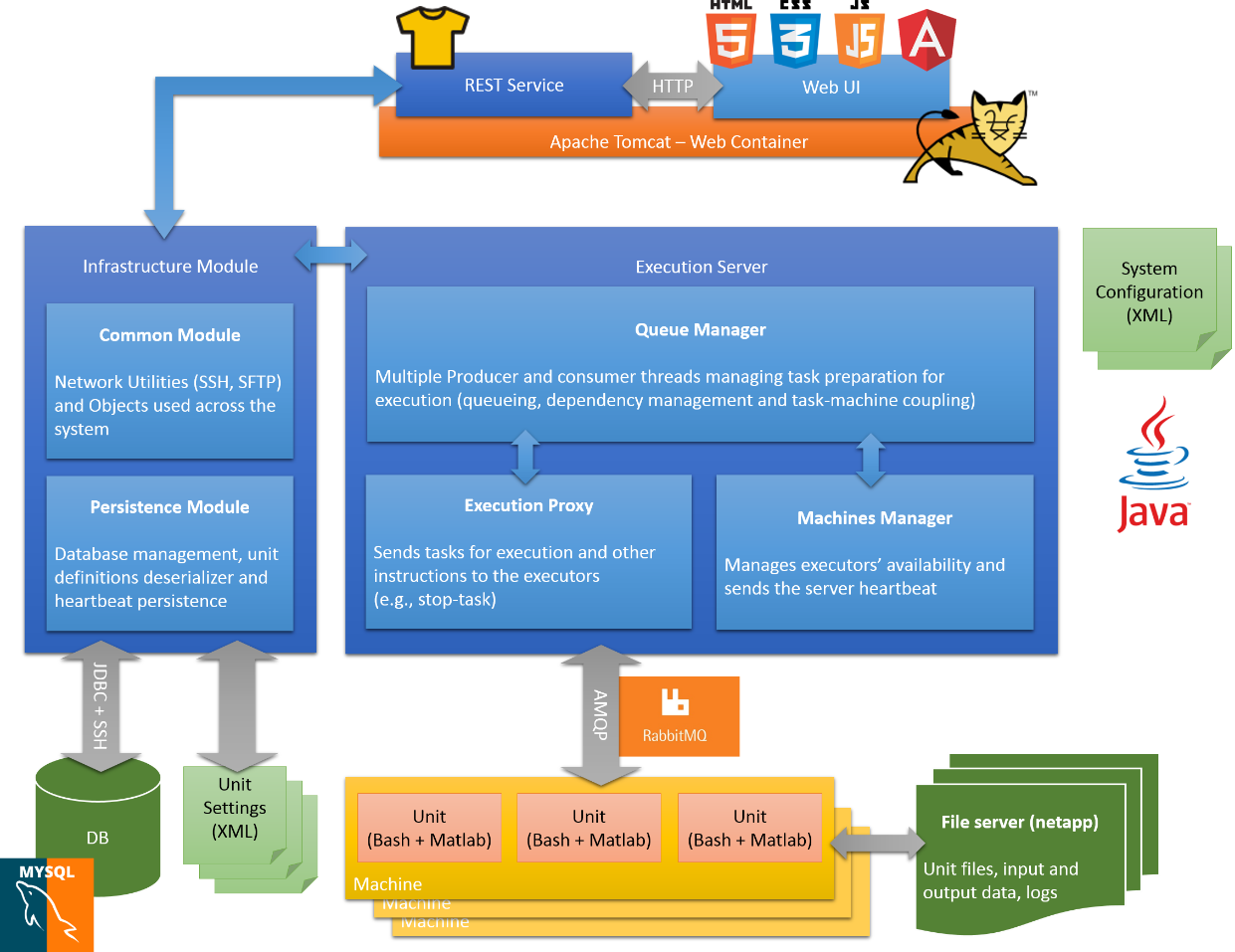
תקציר

בפרויקט זה פיתחנו מערכת לניהול ולהרצת אנליזות עבור מעבדות מחקר בתחום מדעי המוח. כלי זה הופך את תהליך הרצת האנליזות לידידותי למשתמש ומאפשר ביצוע פעולות מתקדמות ללא ידע טכני מעמיק, אשר נדרש טרם פיתוח המערכת. הכלי מאפשר הרצה מקבילית ומבוזרת של אנליזות ללא צורך בהתערבות אנושית, בעוד במצב הנוכחי, כל אנליזה מתוך רצף משימות דורשת הפעלה ומעקב ידניים, וכמו כן, ביזור האנליזות מתבצע בצורה ידנית – דבר הדורש זמן וידע טכני ניכרים.

מקרה הבוחן לפרויקט נבחר להיות אנליזות המורצות על פלטי EEG, אולם המערכת תוכננה להיות כללית וגמישה לסוגי אנליזות מגוונים; במהלך הפיתוח היא נבדקה גם על אנליזות fMRI בסיסיות.

המערכת מורכבת משלושה חלקים עיקריים:

1. שירות HTTP מקושר לממשק משתמש מבוסס טכנולוגיות Web, המאפשר הפעלת סדרות של אנליזות שונות, בקרה אחר האנליזות הרצות והמכונות הזמינות במערכת, כמו גם ביצוע פעולות כמו הוצאת מכונות משימוש, עצירה או הפעלה מחודשת של משימות וכיו"ב.
2. שרת מרכזי האחראי על הקצאת המשאבים, תזמון המשימות ועיקר השמירה על תקינות מסד הנתונים.
3. תהליכי ג'אווה המופעלים על-גבי מכונות שונות במקביל, מריצים בפועל את האנליזות ומדווחים לשרת על תוצאות הרצתן, במקרי הצלחה או כשלון באמצעות Messaging Broker.



איור 1 – דיאגרמת בלוקים וטכנולוגיות בשימוש

תוכן העניינים

1. הקדמה – עמוד 4
2. רקע תיאורטי – עמוד 5
3. מימוש – עמוד 6
   1. מודולים משותפים – עמוד 6
      1. מבני נתונים – עמוד 6
      2. כלי רשת – עמוד 7
   2. השרת המרכזי – עמוד 8
      1. המחלקה הראשית ושער השליחה – עמוד 8
      2. מנהל התור – עמוד 9
      3. מנהל המכונות – עמוד 10
   3. מכונות הקצה – עמוד 10
   4. שירות HTTP – עמוד 12
   5. ממשק המשתמש – עמוד 14
      1. תצוגת שליטה ובקרה – עמוד 15
      2. תצוגת ניהול – עמוד 16
      3. תצוגת יצירת אנליזות ומשימות – עמוד 17
   6. מסד הנתונים – עמוד 17
   7. בנייה, אריזה וניהול גרסאות – עמוד 18
4. סיכום, מסקנות והצעות להמשך – עמוד 19

רשימת איורים

1. דיאגרמת בלוקים וטכנולוגיות בשימוש – עמוד 2
2. תורי ראביט – עמוד 8
3. דיווח תוצאות הרצת משימה לתור ראביט – עמוד 8
4. צילום מסך של תצוגת השו"ב – עמוד 16
5. צילום מסך של תצוגת הניהול – עמוד 16
6. צילום מסך של תצוגת יצירת ה-Flow – עמוד 17

רשימת טבלאות

1. REST API – עמוד 13

# הקדמה

מטרת הפרוייקט הינה פיתוח מערכת אוטומטית ומבוזרת לניהול ולהרצת אנליזות עבור מעבדות מחקר בתחום מדעי המוח, שתאפשר הרצת אנליזות מסוגים שונים – "בלחיצת כפתור".

טרם פיתוח הפרוייקט, כל הרצה של אנליזה דרשה כתיבת סקריפטים מותאמים אישית ומעקב אחר ריצה של כל סקריפט (המהווה חוליה בשרשרת של סקריפטים). הפעלת הסקריפטים נעשית כיום באופן ידני, ועל-כן אי-זמינות של האדם המפעיל סקריפט מסוים גורם לעיכוב בהרצת הבאים בתור. תהליך זה אורך זמן רב, ואינו מנצל בצורה יעילה את כל השרתים הזמינים לעיבוד אנליזות.

בנוסף לתקורה במימד הזמן, המצב הנוכחי דורש מהמשתמשים שימוש בידע טכני ובכלים לא נוחים; אין בידי החוקרים במעבדות כלי גרפי שמנהל ומאפשר בקרה אחר ריצת האנליזות. זו אחת הסיבות המרכזיות לשמן נבנתה המערכת אשר תוצג במסמך זה.

כיום במעבדה נעשה שימוש במערכת BrainVoyager[[1]](#footnote-1), הנותנת מענה חלקי לצרכים הנ"ל. על אף היותה מערכת משוכללת המאפשרת ביצוע אנליזות מתקדמות, היא מוגבלת במידת ההתאמה שלה לצרכי המעבדה. הפערים מגושרים כיום על-ידי הרצת סקריפטים באופן ידני, כאמור. המערכת צפויה לאפשר בעתיד הקרוב הרצה של אנליזות על-גבי מכונות שונות במקביל ואפשרויות מתקדמות נוספות.

עם זאת, בשונה מ-BrainVoyager, הפתרון שהתבקשנו לממש יועד להיות כללי יותר וגמיש, ומסוגל לתמוך בכל סוג אנליזה, ומצד שני, לתת מענה מדוייק לצרכי המעבדה באוניברסיטה. בנוסף, המערכת שתתואר במסמך זה תוכננה להיות בעלת ממשק משתמש פשוט ונוח המאפשר הרצת אנליזות וניהולן ללא הכשרה ארוכה או מורכבת.

בפרוייקט זה תיכננו ומימשנו מערכת שתאפשר:

* **ביזור, הרחבה (Scalability), מקביליות וקיצור זמני ריצה** באמצעות הרצה מסונכרת ומתוזמנת אוטומטית על-גבי מספר מכונות ניתן לשינוי ולשליטה.
* **גמישות** – המערכת כללית ומסוגלת להריץ כל סט של יוניטים – אבני הבניין של אנליזות – ללא חשיבות לסוג המידע עליו הן מופעלות (יהיו אלו פלטים של צילומי fMRI, פלטי EEG וכן הלאה).
* **שימוש חוזר** – עבור כל יוניט, המשתמש נדרש לכתוב את קוד המטלב[[2]](#footnote-2) פעם אחת בלבד. קוד זה יוכל לשמש את אותו משתמש וכן משתמשים נוספים, באנליזות שונות, בעזרת השמת ערכי קלט וערכי הקשר משתנים, המהווים התאמה אישית של האנליזה.
* **ניהול ושליטה** – ניהול מכונות, מעקב אחר אנליזות, וביצוע פעולות לפני הרצתן במכונות, אחריה וגם תוך כדי הריצה. כל היכולות מוטמעות בממשק משתמש נוח, פשוט וזמין מכל מקום ברשת האוניברסיטה.

בתכנון המערכת חקרנו ארכיטקטורות שונות למימוש, שפות תכנות וטכנולוגיות מתאימות וזמינות לשימוש באוניברסיטה. כמו כן, ניסינו להבין את אופי העבודה של החוקרים במעבדה ולהתאים את המערכת לצרכיהם.

# רקע תיאורטי

בעשורים האחרונים הרפואה המודרנית וחקר מערכות הגוף התפתחו בצורה משמעותית, בין היתר הודות לפיתוחים טכנולוגיים פורצי דרך. מחקרים בתחומים מדעיים שונים ובהנדסה הובילו לפיתוח שיטות חדשות של קליטה, ניתוח ועיבוד אותות אלקטרומגנטיים, ויישומם בתחום הרפואה. היישומים כוללים סוגי דימות שונים (טומוגרפיה לסוגיה, fMRI וכיו"ב) ושיטות אלקטרוגרפיה (כגון EEG), המאפשרים קבלת מידע על מבנה ותפקוד המוח, שלא היה זמין עבור חוקרים ורופאים בעבר או שהיה קיים אך לא ברמת הדיוק הנוכחית.

המוח האנושי מורכב מכ-80-90 מיליארד נויורנים [1], מה שהופך אותו לאיבר מורכב ביותר שמבנהו ותפקודו נמצאים בחזית המחקר הרפואי. לאור מבנהו הצפוף והמסועף וכמות האותות החשמליים החולפים בו בהבזקים קצרים ומהירים, כמות המידע שנאסף על-ידי מכשירים הקולטים את אותות המוח היא אדירה ודורשת יכולות ומשאבי עיבוד מתקדמים.

במעבדות המחקר של בית הספר סגול לחקר המוח, מתבצעים ניסויים שונים כחלק ממחקרים בתחום, ונאסף במהלכם מידע רב. מידע זה נאגר בשרתי האוניברסיטה ועובר עיבוד על-ידי תוכנות ייעודיות, כגון BrainVoyager, ועל-ידי קוד שפותח באוניברסיטה, למשל במטלב. עיבוד המידע, לאור נפחו, מגוון סוגיו ושלבי האנליזה הרבים שיש לבצע עליו כדי להגיע לתוצר סופי בעל ערך, הוא תהליך ארוך ומורכב. מכאן נולד הצורך במערכת שתתואר להלן.

# מימוש

בבואנו לממש את המערכת נדרשנו למספר החלטות, אשר עיקרן בחירת טכנולוגיות ותכנון הארכיטקטורה והתקשורת בין הרכיבים השונים. מדרישות המערכת ברור כי נדרשת הפרדה בין קוד שרת מרכזי לבין תהליכים עצמאיים (Processes) אשר ירוצו במקביל על מספר מכונות, וכי על כל תהליך שכזה לתקשר באופן דו-כיווני עם השרת. כמו כן, על-מנת למנוע מצב בו השרת יכפה אילוצים על המשתמשים, כמו ביצוע פעולות משתמש באותה מכונה עליה רץ השרת, החלטנו לאפשר גישה מרוחקת לשרת באמצעות ממשק משתמש בלתי תלוי. ההבנה הראשונה וההחלטה השניה, דרשו תכנון אפיקי תקשורת בין הרכיבים ופתרונות לסנכרון פעולתם.

השפה העיקרית בה הוחלט לפתח את הרכיבים בצד השרת הייתה Java 1.8, ממספר סיבות:

1. מידול התוכנה התאים לתכנון מונחה עצמים, בו Java היא שפה מתקדמת ודומיננטית.
2. שפת Java איננה תלויה במערכת ההפעלה או החומרה עליה היא רצה (Portability) ולכן דורשת פחות אילוצים מצד המעבדות לחקר המוח.
3. התקנה ותפעול נוחים, כולל תמיכה פנים אוניברסיטאית (לדוגמה, בשרתי האוניברסיטה מתוחזקים Servlet Containers מסוג Tomcat, אשר מאפשרים הרצת שירותי HTTP מבוססי Java).

גם ביתר חלקי המערכת השתמשנו בטכנולוגיות המובילות בשוק, כגון AngularJS עבור ממשק משתמש דינאמי ו-RabbitMQ להעברת הודעות אמינה ותומכת נפחים גדולים (Scalability). השימוש שנעשה בהן והסיבות לבחירות אלו יפורטו בהמשך הפרק.

ארכיטקטורת המערכת מוצגת באיור 1 – הסברים על הרכיבים השונים, כפי שמופיעים באיור זה, יובאו בהמשך הפרק. כל הרכיבים פועלים לפי קובץ קונפיגורציה מרכזי, המגדיר את פרטי החיבור למסד הנתונים, את המכונות המחוברות למערכת ונתונים נוספים. על-ידי שינוי הערך בפרמטר ה-Debug בקובץ הקונפיגורציה, ניתן להפעיל את המערכת בתצורת Staging ובמצב זה כל העבודה מול מסד הנתונים מתבצעת על-גבי טבלאות debug ולכן אין לה השפעה על הנתונים השמורים במסד הנתונים המבצעי (Production).

## מודולים משותפים (Infra)

כפי שתואר בתחילת הפרק, המערכת מורכבת ממספר רכיבי Java נפרדים. עם זאת, כחלק ממערכת אחת, הם חולקים באופן טבעי יישויות (Entities) ויכולות משותפים. לשם כך הוגדר מודול התשתית, אשר מורכב משני חלקים: ספריות משותפות (Common) ורכיב שמירת הנתונים (Persistence).

הספריות המשותפות מורכבות ממספר חבילות (Java packages) שהמרכזיות ביניהן הן חבילת מבני הנתונים וחבילת כלי הרשת.

### מבני נתונים

חבילת מבני הנתונים כוללת מחלקות שמגדירות עצמים (Java objects) במודל התוכנה של המערכת, אשר מותאם ליישויות בעולם התוכן. עצמים מרכזיים במערכת:

1. יחידת עיבוד (Unit), להלן יוניט – היוניטים מהווים את אבני הבניין של האנליזה. כל יוניט מהווה שלב אחד בתהליך העיבוד של ה-data. יישות היוניט מכילה את המידע המגדיר יחידת עיבוד – מזהה ייחודי, פרמטרים, תיאור קצר, נתיב קבצי הקלט בהם נמצאים הקבצים לעיבוד ונתיב הפלט בו יש לשמור את הקבצים הנוצרים בתהליך.
2. מכונה (Machine) – יישות המכונה מכילה את המידע הדרוש לניהול המכונות : מזהה ייחודי, כתובת IP, תיאור (hostname), שדה בוליאני השומר את מצב המכונה (האם המכונה רצה), שדה בוליאני המגדיר האם המכונה מוגדרת כחלק ממערך המכונות הזמינות, ושדה מסוג timestamp המחזיק את הזמן האחרון בו שלחה המכונה Heartbeat לשרת.
3. הקשר (Context) – יישות ההקשר מכילה שלושה שדות: שם המחקר, שם הנבדק ושם/מספר הריצה.
4. משימה (Task) – זו היישות המתכללת את המידע הרלוונטי לביצוע משימה של ממש על-ידי מכונה מסוימת. עצם זה מורכב מקישור ל-Flow (יישות המוגדרת במסד הנתונים ומורכבת מהקשר ומרשימה סדורה של משימות), מיחידת עיבוד שהוגדרו עבורה ערכי פרמטרים, ומהקשר.

במערכת כולה העצמים עוברים מספר תווכי תקשורת ואחסון ולכן יש צורך לקודד אותם (Serialization) ולאחזר/לפענח אותם (Deserialization).

תווכי התקשורת כוללים את RabbitMQ, בו העצמים מסוג RabbitData (המורכב מהודעה ומאובייקט Task) עוברים בהודעות כבתים (bytes), ואת התקשורת בין שירות ה-HTTP לבין ממשק המשתמש, בו הקידוד בשימוש הוא JSON (כיוון שפורמט זה מגדיר באופן טבעי אובייקט בשפת JavaScript). הקידוד הראשון מתבצע באופן משתמע (implicit), על-ידי מימוש ממשק Serializable על-ידי רוב היישויות. הקידוד ל-JSON מתבצע באופן ישיר (explicit) על-ידי חבילות קידוד מוכרות כגון Gson של Google.

לצד תווכי התקשורת, גם בתחום האחסון ואחזור הנתונים מהזכרון ומהדיסק, יש צורך בקידוד. גם במקרה זה מתבצעים שני סוגים – קידוד לקבצי XML, שנוגע לקבצי ההגדרה של יוניטים ולקבצי קונפיגורציה, וקידוד לצורך שמירה במסד הנתונים. הקידוד לקבצים מתבצע על-ידי אנוטציות (annotations) ולכן גם הוא implicit. השמירה למסד הנתונים מתבצעת בצורה explicit במיוחד, כיוון שיש לבצע המרה של כל שדה של העצם לשדה מתאים במסד הנתונים ולבסוף לספק תיאור עבור עצם שמתאים להיות חלק מהצהרת SQL.

### כלי רשת

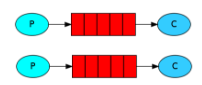
במערכת קיימים מספר רכיבי תקשורת:

1. [[3]](#footnote-3)SFTP – המכונות משתמשות באובייקט ה- SftpManagerלצורך הורדת קבצי קלט מהשרת למכונה, והעלאת קבצי פלט מהמכונה לשרת. ה- SftpManager תומך בנתיבי קלט המכילים ביטויים רגולרים ומנהל את תהליך זיהוי הנתיבים המתאימים בשרת, זיהוי הקבצים המתאימים בתוך הנתיבים הללו, ולבסוף מוריד את הקבצים המתאימים באמצעות פרוטוקול SFTP, להורדת קבצים מאובטחת.
2. RabbitMq (להלן ראביט) – את התקשורת בין השרת המרכזי לבין המכונות מימשנו באמצעות תורי ראביט. הראביט הינו Messaging Broker המותקן במכונות ובשרת ומאפשר ניהול מסודר ובטוח של התקשורת. הודעה הנשלחת על גבי תורי הראביט תגיע ליעדה גם אם בזמן שנשלחה, המכונה או השרת המתאימים אינם רצים. ההודעה תמתין בתור עד אשר המכונה או השרת יעלו ויאזינו לתור.

לכל מכונה קיימים שני תורים משלה, אליהם היא מאזינה בקביעות בשני חוטים (Threads) שונים:

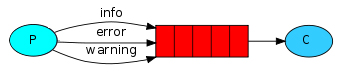
* ubongo\_tasks\_queue – תור לקבלת משימות חדשות לביצוע.
* ubongo\_kill\_requests\_queue - תור לקבלת בקשה לעצירת משימה שרצה על המכונה.

כאשר מגיעה הודעה לאחד מהתורים, המכונה מיד יוצרת חוט חדש שיטפל בבקשה, וממשיכה להמתין להודעות חדשות.



איור 2 – תורי ראביט

לשרת קיים תור אחד לו הוא מאזין, הנקרא ubongo\_tasks\_status\_queue. באמצעות תור זה מדווחות המכונות לשרת על תוצאות הרצת משימות, במקרי הצלחה או כשלון.



איור 3 – דיווח תוצאות הרצת משימה לתור ראביט

1. SSH[[4]](#footnote-4) – מסד הנתונים נמצא בשרת נפרד באוניברסיטה וכדי לבצע אל-מולו שאילתות, אנו משתמשים ב-SHH Tunneling. אנו יוצרים "מנהרה" באמצעות שער (Port) שמזוהה כפנוי, ודרכו אנו מעבירים את התעבורה אל מסד הנתונים וממנו. על-מנת לבצע את חיבור ה-SSH נדרשים פרטי הזדהות ולכן פיתחנו תמיכה בשתי אפשרויות חיבור: באמצעות סיסמא או באמצעות קובץ SSH Key. המשתמש יכול להגדיר בקובץ הקונפיגורציה את אחד מהאמצעים הללו לפי העדפתו. באוניברסיטה קיימת תמיכה ליצירה פשוטה של מפתח לפי הצפנת RSA, היוצרת מפתח פרטי וציבורי. בקובץ הקונפיגורציה על המשתמש להגדיר את הכתובת של המפתח הציבורי בלבד.

## השרת המרכזי

השרת המרכזי הינו תהליך Java מרובה חוטים המורכב ממספר מחלקות: המחלקה הראשית, מנהל התור, מנהל המכונות והשער לשליחת המשימות למכונות.

### המחלקה הראשית ושער השליחה (Execution Server and Execution Proxy)

מחלקת ExecutionServer היא המחלקה המופעלת עם הרצת קובץ ה-JAR של השרת המרכזי. היא מנהלת את תתי-הרכיבים שיתוארו מטה ואת תתי-הרכיבים שתוארו בפרק מודולים משותפים (Infra).

במחלקה זו מופעל החוט אשר מתשאל את מסד הנתונים בהפרשי זמן קבועים אודות בקשות שהתקבלו מהמשתמשים (דרך שירות ה-HTTP; ראו בהמשך), ואחראי לבצען. זו גם המחלקה אשר מאזינה לעדכונים מהמכונות לגבי משימות שהושלמו, נכשלו או הופסקו, דרך תור ייעודי בראביט.

כשלון של מחלקה זו משמעותו קריסה של השרת ולכן היא אחראית גם לתפיסת כל Exception שעלול להיזרק במערכת וטיפול בו. בחלק מהמקרים השרת מבצע ניסיון החלמה מתקלות "סופניות" (Fatal errors), על-ידי עצירה של כל הרכיבים, ניקוי מבני הנתונים שבזכרון, ביצוע פעולות של ניקוי מסד הנתונים ועצירת משימות הרצות במכונות, והפעלה מחודשת של כלל הרכיבים.

שער השליחה היא מחלקה שמשלימה את פעולות השרת ואחראית להעברת משימות להרצה לתור המתאים בראביט וכן לשליחה של פקודות לעצירת משימות, גם כן דרך הראביט.

### מנהל התור (Queue Manager)

מנהל התור הוא הרכיב המתוחכם והמורכב ביותר בשרת המרכזי, ללא כל תחרות. הוא מהווה למעשה "שוטר התנועה" המפנה משימות להרצה על מכונות, מעכב משימות התלויות במשימות שטרם הסתיימו ומעדכן משימות בהתאם ליכולת שלהן להיות מורצות ובהתאם למצבן של משימות אחרות באותו Flow.

אמנם אנו מעריכים כי המערכת אינה צפויה להתמודד עם עומסים כבדים בתחילת דרכה, אך מנהל התור תוכנן להיות סקלבילי (Scalable) ועמיד (Robust) ומסוגל לטפל במקביל במשימות רבות. המנהל בנוי לפי תבנית עיצוב (Design Pattern) קלאסית של Producer-Consumer [3]; כלומר, ישנו תור עליו פועלים חוטים מפיקים, אשר מכניסים אליו משימות, וחוטים צורכים אשר מוציאים ממנו משימות. בתצורה הנוכחית החלטנו להשתמש בחוט מפיק אחד, אבל בהתאם לשימוש במערכת ניתן להגדיר עבודה עם מספר חוטים במקביל.

השרת המרכזי פונה למנהל התור בכל פעם שהוא מריץ Flow, כדי לדווח לו שהוא צפוי לראות משימות חדשות. במקרה זה חוט מפיק מתעורר ומושך ממסד הנתונים את המשימות החדשות. לחוטים המפיקים יש גם מגבלת זמן לשינה ואם לא קיבלו בקשה במשך זמן רב הם מתעוררים כדי לוודא שלא "נרדמו בשמירה". כאשר החוט המפיק מזהה משימות חדשות, הוא ראשית בודק האם ניתן להריץ אותן. לשם כך, מנהל התור מתחזק מספר מבני נתונים:

1. מילון הממפה בין TaskKey לבין DependencyKey. ה-TaskKey הוא מפתח המשותף לכל המשימות תחת אותו Flow החולקות גם את אותו מספר סידורי. כלומר, אלו משימות אשר ניתנות להרצה במקביל ואינן תלויות זו בזו אך הן תלויות באותן משימות קודמות ומשימות עם מספר סידורי גבוה יותר תלוי בהשלמתן. ה-DependencyKey הוא מפתח שמאפשר חיפוש ב-DependencyMap.
2. DependencyMap – מילון הממפה בין DependencyKey לבין קבוצה של משימות שתלויות במשימות המתאימות ל-DependencyKey הנתון. זהו תיאור מבלבל, אולם בפשטות: בהינתן משימה א' שהסתיימה או נכשלה, למשל, ניתן לחפש באמצעותה במילון הראשון ולקבל מפתח. המפתח הזה מאפשר חיפוש ב- DependencyMapכדי לקבל את קבוצת כל המשימות, שתלויות בהשלמת משימה א' ואולי במשימות נוספות. כאשר א' היא המשימה האחרונה שמונעת מהמשימות בקבוצה להיות מורצות, וא' הסתיימה בהצלחה, ניתן להריץ את המשימות הללו.

הצורך בשני מבני הנתונים נובע מהעיצוב של שפת Java, בה לא ניתן להשתמש במפתח למילון שניתן לשינוי (Mutable), לכן לא יכלנו להשתמש בקבוצת המשימות המעכבות כמפתח למילון שהערכים בו הם קבוצות המשימות התלויות.

1. קבוצת משימות לביטול – קבוצה אשר מוכנסת אליה משימה לפני שהיא נשלחת לביטול במסד הנתונים וכאשר מסתיים תהליך הביטול היא יוצאת מהקבוצה. מבנה נתונים זה נחוץ כדי למנוע מצב בו מנהל התור מצד אחד מבקש לבטל משימה ומצד שני חוט מפיק שלו בינתיים מושך אותה ממסד הנתונים ומתחיל לטפל בה לקראת הרצה. במקרה זה היה עלול להיווצר מצב בו המשימה כביכול מבוטלת במסד הנתונים בעוד היא למעשה רצה על אחת המכונות.

באמצעות המבנים לעיל, החוט המפיק מנהל את התלויות בין משימות ונמנע מלהריץ משימות שיש לבטל. אולם בכך לא תמו תפקידיו; משימות הנמצאות במסד הנתונים עשויות לכלול נתיבים תלויי הקשר, בהם יש "לשתול" נתונים, כמו שם נבדק או ריצה של ניסוי. במידה והחוט המפיק מזהה משימה כזו, הוא מחליף אותה במשימות נפרדות שכל אחת מהן רצה על נתוני הקשר בודדים וחד-משמעיים (ללא ביטויים רגולרים).

בסיום כל תהליכי הניהול שמבצע החוט המפיק הוא מכניס לתור את המשימות להרצה ומעיר חוטים צורכים. חוטים אלו גם הם עושים שימוש במבני הנתונים שהוזכרו לעיל כדי לוודא שניתן לשלוח את המשימות להרצה. במידה והמשימות ראויות להרצה, החוט המפיק פונה למנהל המכונות בבקשה למכונה פנויה, מקצה אותה למשימה ושולח להרצה. בכל שלב בתהליך, החוטים מעדכנים את מסד הנתונים כדי לשמור על עדכניות ונכונות הנתונים.

כאשר נשלמת הרצת משימה היא מדווחת לשרת המרכזי ומגיעה למנהל התור בשנית. הפעם, המנהל מעדכן את מצבה במסד הנתונים והיא משמשת לבדיקה במבני הנתונים שתוארו לעיל, האם סיומה מאפשר הרצה של משימות נוספות. במקרים אחרים, משימות שנכשלו או הופסקו עוברות תהליך דומה וגורמות למשימות הבאות בתור להיות מעוכבות (On Hold).

### מנהל המכונות (Machines Manager)

מנהל המכונות הוא הרכיב אותו מתשאל מנהל התור כאשר הוא רוצה לשלוח משימה לביצוע. האחרון שולח בקשה למכונה זמינה ומנהל המכונות בודק אילו מכונות פעילות ומחוברות (כלומר, שלחו סיגנאל Heartbeat בזמן האחרון, כפי שיוסבר עוד בפרק על ממשק המשתמש). מתוכן הוא מקצה מכונה אחת, לפי מונה ציקלי (Round Robin).

בהתאם לעיסוקו של רכיב זה במעקב אחר חיבוריות המכונות באמצעות Heartbeat, הוא קיבל על עצמו אחריות לשליחת ה-Heartbeat מצד השרת המרכזי בהפרשי זמן קבועים. סיגנאל זה משוקף בממשק המשתמש ויוסבר בפרק העוסק בו.

## מכונות הקצה

מכונות הקצה מקבלות משימות מהשרת המרכזי ומבצעות אותן. ישנם שני סוגי משימות:

1. הרצה של משימה.
2. עצירה של משימה שרצה במכונה.

השרת מעביר למכונות את המשימות באמצעות תורי RabbitMQ. כאשר כל משימה נשלחת כאובייקט מסוג RabbitData המכיל את השדות הבאים:

1. Message – מזהה של הפעולה לביצוע. כלומר, הרצה או עצירה של משימה.
2. Task – אובייקט משימה, המכיל את כל המידע הנדרש לביצועה.

כאשר המכונה מקבלת הודעה באמצעות תור הראביט, היא מעבדת תחילה את ההודעה בכדי למיין אותה לסוגי הפעולות האפשריים, ומחלצת ממנה את אובייקט המשימה. הטיפול בכל הודעה מתבצע על-ידי חוט חדש, על-מנת לאפשר מקביליות מלאה ולא לפגוע בזמינות המערכת. החוט המטפל בהודעה, שולח אותה לפונקציה המתאימה לפי סוג הפעולה הדרוש.

1. עבור בקשה מסוג עצירת משימה רצה:

המכונה תשלח Interrupt לחוט המתאים, המריץ את המשימה (אותו ניתן לאתר לפי מזהה המשימה). כאשר ה-Interrupt יקלט במערכת תתבצע עצירה מסודרת של המשימה הנדרשת וניקוי הקבצים הזמניים שנוצרו במערכת. בסיום התהליך תישלח הודעת עדכון לשרת.

1. עבור בקשה מסוג הרצת משימה חדשה:

בתהליך הרצת בקשה חדשה המכונה משתמשת בקבצי ה-Units. היוניטים מהווים את אבני הבניין של האנליזה; כל יוניט מהווה שלב אחד בתהליך עיבוד המידע והוא מורכב משלושה קבצים:

* Unit.xml – קובץ הקונפיגורציה של היוניט, המכיל:
  + נתיב קבצי הקלט בהם נמצאים הקבצים לעיבוד. נתיב זה יכול להיות קבוע, להכיל ביטויים רגולרים (Regex) וכן לכלול שדות השייכים להקשר (Context), שיתעדכנו בערך מתאים על-ידי המשתמש בזמן הרצת האנליזה.
  + נתיב קבצי הפלט אליו יש לשמור את הקבצים הנוצרים בתהליך העיבוד. נתיב זה יכול להיות קבוע, וכן לכלול שדות שיתעדכנו בערך מתאים על-ידי המשתמש בזמן הרצת האנליזה, כך שיתאימו לאנליזות שונות – שדות אלו הם: שם המחקר, שם הנבדק ושם/מספר הריצה. מילוי ערכי ההקשר מתבצע בשרת.
  + פרמטרים המהווים קלטים של היוניט. לכל פרמטר יש לספק: שם תצוגה, אשר ישמש להצגה נוחה של היוניט בממשק המשתמש; שם הפרמטר כפי שיופיע בקוד המטלב שיווצר באופן אוטומטי; וערך ברירת מחדל, אשר מוזן לפרמטר בכל זמן שלא קובעים ערך אחר באמצעות ממשק המשתמש – שדה זה אינו שדה חובה.
* Unit.m – קובץ המטלב של היוניט. קובץ זה מכיל את הקוד, האחראי על תהליך העיבוד המתבצע על ידי היוניט. זהו קוד כללי, שאינו מכיל ערכים של משתנים, נתיב קלט ונתיב פלט – מאחר ואלו מתעדכנים בזמן ריצה בהתאם לאנליזה.
* Unit.sh – קובץ סקריפט הרצה (Bash) אשר יוצר קבצי מטלב בזמן ריצה ומריץ אותם. ניתן לייצר אותו באופן ידני, כך שקיימת גמישות מלאה לסקריפטים שונים ומגוונים. בנוסף ניתן לייצר אותו באופן אוטומטי באמצעות ממשק המשתמש, בהינתן קובץ המטלב וקובץ הקונפיגורציה של היוניט, כאשר ניתן להגדיר בקונפיגורציית המכונה תלויות שיש להוסיף לקבצי ה-Bash. סוג הסקריפט האוטומטי הנתמך במערכת כיום תואם את הדרישות הקיימות, אך ניתן לתמוך ביצירת קבצי Bash מסוגים נוספים בצורה קלה יחסית.

לצורך הרצת משימה, המכונה תחילה משתמשת בנתיב הקלט שהתקבל באובייקט המשימה ומורידה את כל קבצי הקלט הנדרשים לתיקייה זמנית, שתישמר במכונה למהלך העיבוד, בנתיב המוגדר בקונפיגרוציה כ-machineWorkspace. נתיב זה נבנה על-ידי השרת מנתיב הקלט של היוניט, בשילוב עם משתני ההקשר של האנליזה. כפי שצויין, נתיב זה עשוי לכלול ביטויים רגולרים, ובזמן הורדת קבצי הקלט מהשרת, המכונה עוברת על כל הנתיבים המתאימים לביטויים אלו, ומורידה רק את הקבצים המתאימים לביטוי הרגולרי של שם הקובץ (במידה ומוגדר כזה). תהליך ההורדה מתבצע באמצעות פרוטוקול SFTP בכדי לאפשר הורדת קבצים משרתים מרוחקים.

לאחר שלב זה, כאשר כל הקבצים שיש לעבד נמצאים בתיקייה הזמנית במכונה – המכונה מריצה את קובץ ה-Bash המתאים של היוניט עם הפרמטרים שהוגדרו בקובץ הקונפיגורציה של היוניט (כאשר להם ערכים דיפולטיים או ערכים שהוזנו על-ידי המשתמש בממשק).

קובץ ה-Bash מקבל את הפרמטרים הללו, ובנוסף מקבל פרמטרים נוספים: הנתיב הזמני של קבצי הקלט במכונה, נתיב זמני במכונה לקבצי הפלט שיווצרו במהלך העיבוד ואת מזהה המשימה. הסקריפט יוצר בזמן ריצה קבצי מטלב (קובץ מטלב לכל קובץ קלט), להם מועברים הפרמטרים הנ"ל וכן שורות העיבוד מקובץ ה-Unit.m המכילות את לוגיקת העיבוד.

את קובץ המטלב שנוצר מריץ סקריפט ה-Bash, כאשר הקובץ וגם לוג ההרצה נשמרים במכונה בנתיב ה-bashTmp הנמצא בתיקיית Units. שמות קבצי המטלב והלוגים שנוצרו מכילים את מזהה המשימה.

אם שלב מסויים בתהליך כשל – המכונה עוצרת, מנקה את קבצי הקלט והפלט הזמניים שיצרה ומודיעה לשרת שהמשימה נכשלה.

לאחר שלב זה, המכונה בודקת האם נתיב קבצי הפלט במכונה מכיל קבצים. אם לא – המכונה מודיעה לשרת שהמשימה נכשלה; אחרת – המכונה מתחילה בתהליך העלאת קבצי הפלט לתיקייה המתאימה בשרת כפי שמוגדר בקובץ הקונפיגורציה של היוניט – ומודיעה לשרת שהמשימה הסתיימה בהצלחה.

גם תהליך ההעלאה לשרת מתבצע באמצעות SFTP.

בסיום התהליך, נתיבי הקלט והפלט הזמניים במכונה נמחקים, ואילו קבצי המטלב שנוצרו במכונה נשארים בנתיב בו הם נוצרו בכדי לאפשר למשתמש להשתמש בהם לצורך בדיקות והבנה מלאה של התהליך שהתבצע.

## שירות HTTP

עד כה תיארנו את החלק הביצועי של המערכת, אולם לא ברור כיצד המערכת מקבלת הוראות ממשתמשים חיצוניים. כדי לקבל קלט מהמשתמש ישנן מספר אפשרויות, כאשר אפשרות בסיסית אחת היא לפתח ממשק (גרפי או טקסטואלי) ישיר בין השרת לבין המשתמש. החסרון בממשק כזה הוא האילוץ הלוקאלי – המשתמש נדרש לעבוד על אותו מחשב עליו רץ השרת. ניתן לחשוב על פתרונות דומים שיאפשרו ממשק משתמש שדורש הפעלה על-גבי אותה רשת מחשבים, והעברת קלט ופלט באמצעות זכרון משותף או משאבים אחרים, בעלי מורכבות גבוהה יחסית ועדיין מגבילה.

על-מנת להפריד בין שרת ה-Backend לבין המשתמש בחרנו ליצור Web Application אשר יוכל לבצע שאילתות אל-מול מסד הנתונים ולהעביר באמצעותו בקשות של המשתמש לשרת בתור בקשות ייעודי למשימה. בשיטה זו, שני הרכיבים בלתי תלויים – בזמן שממשק המשתמש אינו זמין, השרת ממשיך לעבד משימות קודמות, ובמידה והשרת הופסק מסיבות כלשהן, ממשק המשתמש זמין וניתן להמשיך לקבל מידע על המערכת ולהעביר הוראות שיבוצעו ברגע שהשרת יהיה זמין שוב.

בהמשך הפרק נפרט אודות ממשק המשתמש הגרפי, אולם גם לו יש ממשק המקשר אותו לתהליכי העיבוד בצד השרת – ה-REST Service. [[5]](#footnote-5)REST [2] הוא סגנון תוכנה אריכטקטוני למימוש שירותי רשת המבוסס לרוב על תקשורת HTTP בין הלקוח (ממשק המשתמש) לבין צד-השרת שמבצע עיבוד על בקשותיו או הוראותיו. במערכת שלנו השתמשנו ב-Jersey, מימוש פופולארי לממשק [[6]](#footnote-6)JAX-RS.

השירות כולל את המתודות המפורטות בטבלה 1 – REST API. ממשק המשתמש עושה שימוש בכל המתודות הללו אולם ניתן לשלוח בקשות והוראות אלה גם באמצעות כלים מתאימים, ללא צורך בממשק משתמש (את בקשות ה-GET ניתן לשלוח באמצעות דפדפן ואת בקשות ה-POST ניתן לשלוח באמצעות כלים ייעודים, כגון Postman[[7]](#footnote-7)). הסבר על עמודות הטבלה:

1. סוג בקשת ה-HTTP; אנו השתמשנו ב-GET וב-POST בלבד.
2. כתובת ה-URI היחסית לביצוע בקשה זו. ניתן לשלוח בקשות אלו באמצעות שרשור הכתובת היחסית ל-URL הבסיס בו נמצא בסיס ה-API[[8]](#footnote-8). בסוגריים מסולסלים רשומים פרמטרים המועברים ב-URI עצמו.
3. הסבר קצר אודות פעולת המתודה, פרמטרים המועברים למתודה באמצעות חלק השאילתא של הכתובת (לדוגמה פרמטר activate שמועבר כך: /machines/2?activate=true) ותוכן הבקשה, אם נדרש.

טבלה 1 – REST API

|  |  |  |
| --- | --- | --- |
| **Description** | **URI** | **Method** |
| Returns API version – good test that the API is responsive | /version | GET |
| Returns details of all the machines | /machines | GET |
| Activates or deactivates a machine depending on a query parameter (activate=true or deactivate=true) | /machines/{id} | POST |
| Returns the names of all stored analyses up to some limit, where the name and limit are passed as query parameters (limit is not mandatory) | /analyses | GET |
| Saves the analysis passed as JSON in the request’s body | /analyses | POST |
| Returns the unit details of the analysis identified by name | /analyses/{name} | GET |
| Returns complete details of all stored flows up to some limit, where the limit is passed as a query parameter (not mandatory) | /flows | GET |
| Saves a flow based on the JSON passed in the request’s body and returns the ID of the created flow | /flows | POST |
| Receives an action as a query parameter (run or cancel) and executes it on the requested flow | /flows/{id} | POST |
| Returns all tasks in the database up to some limit passed as a query parameter (not mandatory) | /flows/all/tasks | GET |
| Returns all tasks of a specific flow up to some limit passed as a query parameter (not mandatory) | /flows/{id}/tasks | GET |
| Receives an action as a query parameter (resume, cancel or stop) and executes it on the requested task | /flows/{id}/tasks/{id} | POST |
| Returns all units as defined by the unit XML files | /units | GET |
| Generates a bash script for a specific unit | /units/{id} | POST |
| Returns all requests up to some limit and created after some point in time (both are query parameters) | /requests | GET |

## ממשק המשתמש

בשנים האחרונות יותר ויותר יצרניות תוכנה מעדיפות לעבור למודל של מתן שירותים באמצעות דפדפנים, על-פני התקנה של אפליקציות על-גבי המחשב של הלקוח. תהליך זה מתרחש ממספר סיבות, כמו שליטה רבה יותר בעדכון גרסאות, פשטות התקנה ותחזוקה ועוד. מודל זה מאפשר לספק תוכנה כשירות במקום כמוצר ועל כן מכונה SaaS[[9]](#footnote-9).

בחרנו לבצע שימוש בממשק המופעל דרך הדפדפן, על-מנת לאפשר שימוש בתוכנה מכל מקום (מתוך רשת האוניברסיטה) וללא צורך בהתקנה מיוחדת. הטכנולוגיות בהן עשינו שימוש לשם כך הינן:

1. שפת HTML – השלד הויזואלי של ממשק המשתמש.
2. שפת CSS – עיצוב האלמנטים בעמודים. לדוגמה, שליטה בצבעים, מיקום אלמנטים ומימדיהם.
3. תשתית AngularJS – תשתית תוכנה העושה שימוש בשפת JavaScript ומתוחזקת בידי Google. אנגולר מאפשרת כתיבה של עמודי אינטרנט דינאמיים וביצוע פעולות כמו קריאות תקשורת מול שירות ה-REST ושינוי ממשק המשתמש בהתאם לתשובות השירות או לפעולות המשתמש.

חלק זה של המערכת כולל מודול (Module) אנגולר מרכזי שבו מוגדרים הנתיבים של האפליקציה ומחלק אותה לשלוש תצוגות (Views) כפי שיפורטו בפרק זה. כמו כן, כתבנו שירות התראות (Notification Service) חוצה תצוגות לצורך הצגת נוטיפיקציות בראש העמוד כאשר נרשמה בקשה חדשה (Request) במערכת (לדוגמה, בקשה לעצירת משימה או לביטול Flow) והמשתמש אינו צופה בתצוגת הניהול, המציגה מידע לגבי בקשות במערכת.

מושג מרכזי באנגולר הוא ה-Directive – בפשטות, מדובר בדרך של המפתח לשנות את ההתנהגות של תגיות ה-HTML (למעשה לא רק לתגיות, אלא גם attributes ואפילו הערות) וכן להגדיר עצמים חדשים בשפה. בפיתוח המוצר שלנו השתמשנו מן הסתם בדירקטיבות אנגולר מהספריה הבסיסית (angular.js), אולם עשינו גם שימוש נרחב בדירקטיבות ובשירותים של ספריית ui-grid[[10]](#footnote-10) ואיפשרו להציג טבלאות דינאמיות עם אפשרויות מתקדמות כמו סידור מחדש של שורות ועמודות, הוספה/הסרה של שורות, בחירה וכיו"ב.

מושג מרכזי נוסף הוא ה-Controller. לכל תצוגה בממשק הגדרנו שניים-שלושה קוטרולרים. קונטרולר היא פונקציה אשר מקושרת לאזור מסוים בשלד העמוד ושולטת על המתרחש באזור זה, לכן הגדרנו קונטרולר לשני צדי המסך בכל תצוגה ובדרך כלל גם לתצוגה עצמה. את הפונקציה רושמים במודול וכך אנגולר יודע להפעיל קונטרולר מתאים כאשר עוברים בין תצוגות שונות. פונקציות אלו מבצעות פעולות שונות – חלקן באות לידי ביטוי בצורה ויזואלית במסך המשתמש וחלקן דואגות לעדכן או לתשאל את מסד הנתונים דרך שירות ה-HTTP (פנייה למשאבים שהוגדרו בטבלה 1 – REST API).

להלן נתאר את הפעולות שניתן לבצע בכל אחת מהתצוגות בממשק המשתמש.

### תצוגת שליטה ובקרה (Dashboard View)

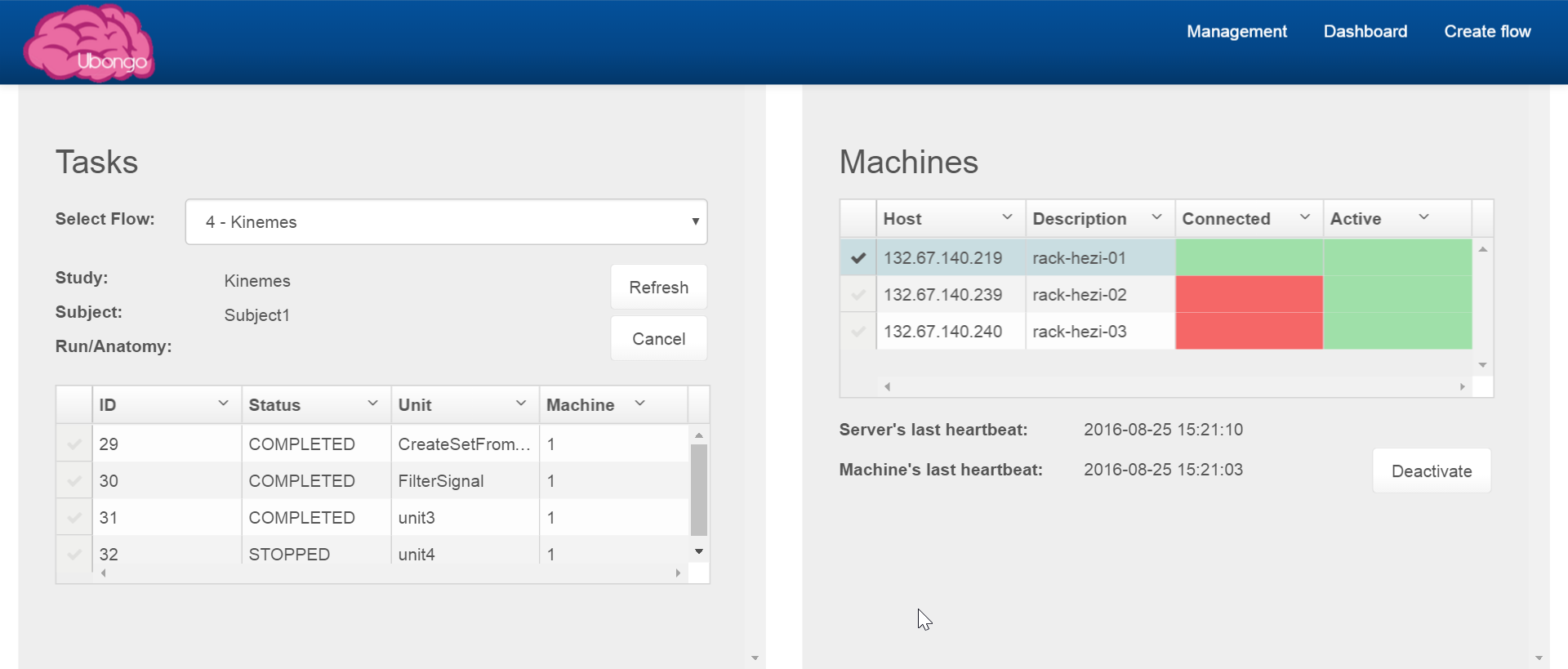
בתצוגה זו ניתנת למשתמש אפשרות לשליטה על המכונות והמשימות הרצות במערכת ובקרה על התקדמותן.

בצדו הימני של המסך מוצגות המכונות הרשומות במסד הנתונים, ונטענות מתוך קובץ הקונפיגורציה של המערכת. כל המכונות המחוברות, ובכללותן השרת המרכזי, שומרות למסד הנתונים אחת לדקה סינגאל שנקרא Heartbeat ומטרתו לעקוב אחר מצבן. במידה והסיגנאל התקבל בטווח זמן של שתי הדקות האחרונות, הממשק משקף למשתמש כי המכונה מחוברת וניתנת למשתמש גם האפשרות לראות מתי לאחרונה נשלח הסיגנאל. בנוסף לצפייה במצב המכונות, המשתמש יכול ממסך זה להפוך מכונה לא פעילה לפעילה ולהיפך – המשמעות היא שהמכונה תהיה זמינה לצורך הרצת משימות, כאשר היא מחוברת.

בצדו השמאלי של המסך, ניתנת למשתמש האפשרות לצפות במצב כל המשימות במערכת, בחלוקה ל-Flows. תצוגה זו מאפשרת לצפות בהקשר של ה-Flow כולו ולבטל אותו בלחיצת כפתור. מתחת לנתוני ההקשר, המסך מציג את רשימת המשימות של ה-Flow עם פירוט אודות מצבן, המכונות עליהן הן רצו או רצות וכו'.

בבחירת אחת המשימות מופיעה טבלה נוספת הנותנת מבט מעמיק יותר על המשימה עם פרטים כגון נתיבי הקלט והפלט, הפרמטרים של היוניט שבבסיס המשימה, ההקשר של המשימה (יתכן כי ה-Flow מופעל למשל על כל הנבדקים במחקר מסוים, ובמקרה זה תיווצרה משימות כמעט זהות, הפועלות במקביל על נבדקים שונים ולכן לכל משימה הקשר שונה) ועוד.

כאשר אחת המשימות מסומנת, למשתמש ניתנות אפשרויות לביצוע פעולות נקודתיות על המשימה, במקום על ה-Flow כולו. הפעולות אפשריות בהתאם למצב המשימה – משימות רצות ניתנות לעצירה, משימות הממתינות להרצה ניתנות לביטול, ומשימות שלא הושלמו בהצלחה ניתנות להפעלה מחדש.



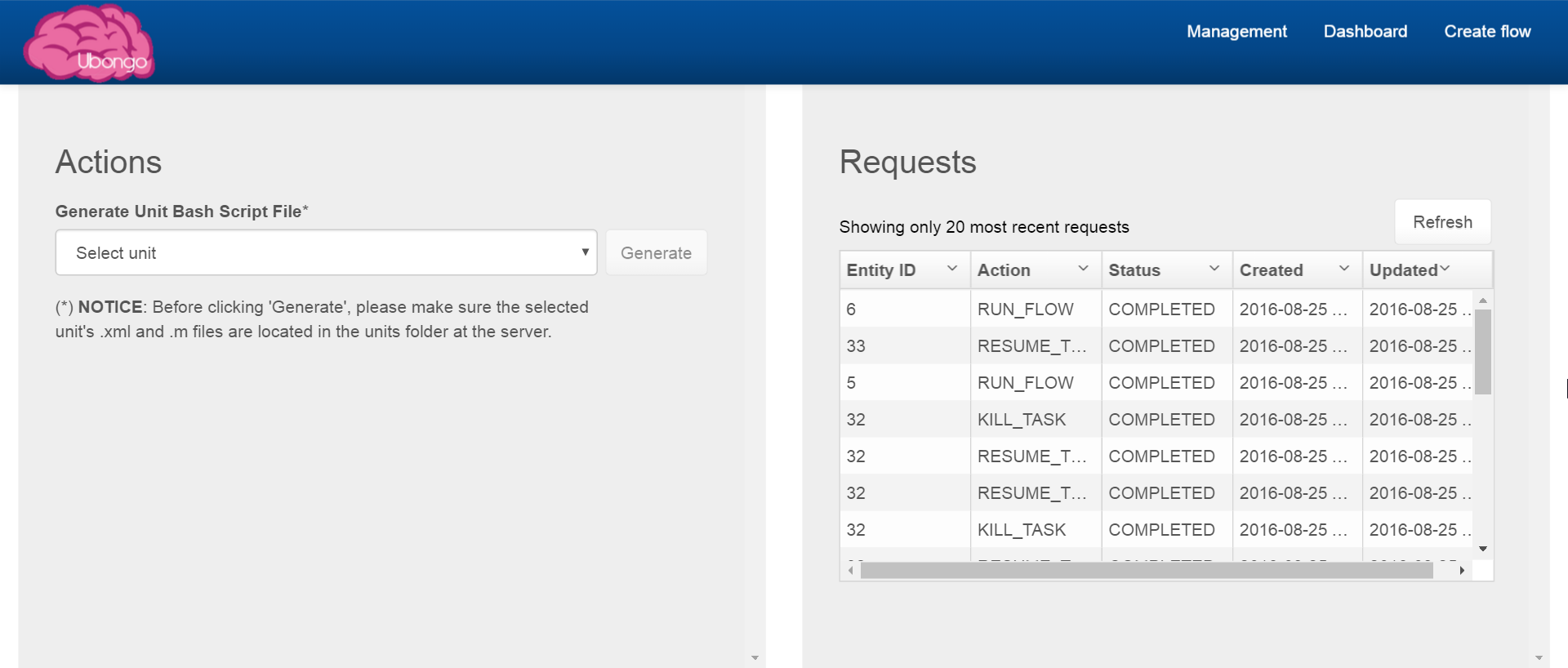
איור 4 – צילום מסך של תצוגת השו"ב

### תצוגת ניהול (Management View)

תצוגת הניהול גם היא מחולקת לשני חלקים: צפיה בבקשות שנשלחו לשרת, ויצירת קבצי bash עבור יוניטים.

בצדו הימני של המסך מוצגות הבקשות שנרשמו במערכת ומפורט מתי נוצרו, מתי עודכנו לאחרונה ומה מצבן הנוכחי. כיוון שהשרת המרכזי מנותק משירות ה-REST, כפי שהסברנו לעיל, חלק מהפעולות אינן יכולות לצאת לפועל על-ידי השירות עצמו ועליהן להיות מופעלות על-ידי השרת (למשל עצירת משימה עשויה להשפיע על משימות אחרות באותו Flow ולכן דורשת סנכרון וניהול זהיר). פעולות אלה נרשמות למסד הנתונים כבקשות ומסך זה מאפשר מעקב אחר הטיפול לו הן זוכות בשרת המרכזי.

בצדה השמאלי של התצוגה המשתמש יכול לבחור יוניט, עבורו קיים קובץ קונפיגורציה (XML) וקובץ Matlab ולייצר עבורו באופן אוטומטי קובץ הרצה לסביבת יוניקס (Bash script).



איור 5 – צילום מסך של תצוגת הניהול

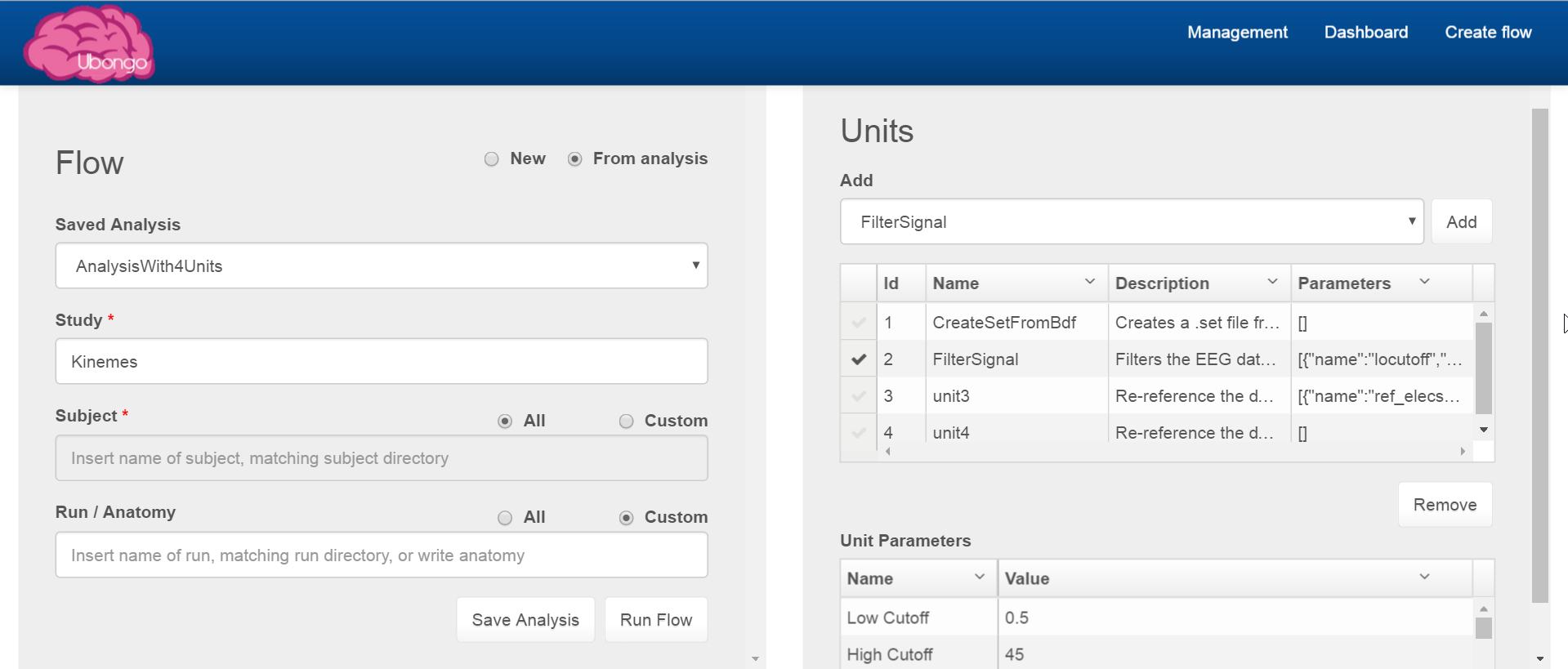
### תצוגת יצירת אנליזות ומשימות (Flow Creation View)

מסך יצירת ה-Flow מחולק כרגיל לשניים, אולם הפעם שני הצדדים פועלים יחדיו ונשלטים על-ידי קונטרולר משותף שמבצע את שליחת ה-Flow לשירות ה-HTTP. בחלק השמאלי של המסך המשתמש או המשתמשת מגדירים נתוני הקשר עבור ה-Flow ובצד הימני בוחרים יוניטים להרצה.

בבחירת נתוני ההקשר, ניתנת למשתמשים אפשרות להזין שמות מחקרים, נבדקים וריצות, כפי ששמורים בעץ התיקיות בשרת ובמכונות. במידה וברצונם להריץ את ה-Flow על כל הנבדקים ו/או כל הריצות עבור נבדק, ניתנת להם אפשרות לסמן זאת בממשק; בחירה זו תגרום לשרת ליצור משימות נפרדות לנבדקים/ריצות השונים באופן אוטומטי (ראו את הפרק על מנהל התור).

בצד ימין של המסך המשתמשים מגדירים את היוניטים אשר ירוצו ב-Flow. מסך זה מאפשר לבחור יוניט מתוך רשימה, להוסיף ולהסיר יוניטים, לסדר אותם מחדש, וכמובן להגדיר את ערכי הפרמטרים של כל יוניט. במידה והפרמטרים תלויים בנתוני הקשר, אלו ישתלו באופן אוטומטי בשרת טרם הרצת המשימות.

עבור מחקרים מסוג מסוים, אנו צופים כי המשתמשים ישתמשו באנליזה (רצף יוניטים מסוים) דומה פעמים רבות, עד כדי הבדלים בנתוני ההקשר ואולי בערכים שינתנו לפרמטרים של היוניטים. מסיבה זו המשתמש יכול להגדיר שם לאנליזה ולשמור אותה לשימוש עתידי. בבואה להשתמש במערכת יכולה משתמשת לטעון אנליזה שמורה ממסך זה, כך ששלב בחירת היוניטים ייחסך עבורה בבואה להריץ Flow חדש.



איור 6 – צילום מסך של תצוגת יצירת אנליזות ומשימות

## מסד הנתונים

בפרויקט עשינו שימוש במסד נתונים רלציוני מסוג MySQL. הוא מהווה רכיב מרכזי במערכת ומשמש לשני צרכים: לשמירה ואחזור נתונים, ולצורכי תקשורת בין רכיבים שונים.

הסכימה בה נעשה שימוש במערכת מורכבת ממספר טבלאות שיפורטו להלן. לכל טבלה ישנם שני עותקים, כאשר עותק אחד משמש ביום-יום (Production) ועותק שני הוא עותק לצורכי בדיקות וטיפול בתקלות (Staging).

1. Flows – שמירה של מזהה Flow, נתוני ההקשר שלו (מחקר, נבדק/ים, ריצה/ות) ומצבו.
2. Tasks – שמירת נתוני המשימות. שדות הטבלה:

* קישור על-ידי Foreign Key למזהה Flow המתאים, כאשר הקישור הוא חד-ערכי (משימות רבות ל-Flow ו-Flow יחיד למשימה).
* סטאטוס – ערך זה משפיע על הפעולות המתאפשרות לביצוע על המשימה.
* מספר סידורי ב-Flow – מאפשר להגדיר תלויות בין משימות ולאכוף אותן במנהל התור.
* מזהה יוניט וערכים לפרמטרים שלו, הנשמרים כ-Blob (כלומר, קישור לטקסט חיצוני לטבלה).
* נתוני הקשר. כאמור, אלו עשויים להיות שונים מנתוני ההקשר של Flow במקרה שההקשר של ה-Flow כולל, לדוגמה, נבדקים רבים, ואילו משימה פועלת על נבדק מסוים.
* מזהה מכונה במידה והשרת כבר הקצה כזו למשימה. ערך זה מאפשר לעקוב אחר המשימה ולעצור אותה במידת הצורך.
* חותמות זמן שונות (זמן הכנסה, זמן עדכון אחרון, זמן תחילת ריצה וזמן סיום).

בטבלה זו הוגדרו טריגרים שמעדכנים את חותמות הזמן בהתאם לעדכון מצב המשימה (למשל, כאשר המצב משתנה ל-Processing, נרשמת חותמת הזמן של תחילת הריצה).

1. Units – היוניטים עצמם שמורים בקבצי קונפיגורציה וסקריפטים להרצה. מטרת הטבלה היא למעשה לשמור אנליזות. היא מקשרת בין שם אנליזה שניתן על-ידי המשתמש, לבין היוניטים שהיא כוללת.
2. Machines – המכונות, בדומה ליוניטים, גם הן מוגדרות בקובץ קונפיגורציה ולכן לא ברור מיד מדוע נדרשת להם טבלה במסד הנתונים. טבלה זו משמשת למטרות תקשורת – אחת לזמן מוגדר מראש (דקה כיום) כל מכונה ושרת שולחים סיגנאל Heartbeat כדי לדווח כי הם מחוברים. שליחת הסיגנאל היא למעשה שמירה למסד הנתונים את הערך 1 בשדה Connected. שמירה זו מפעילה טריגר שרושם במסד הנתונים את חותמת הזמן של קבלת הסיגנאל – חותמת המשוקפת למשתמש בממשק ומשמשת את מנהל המכונות בבחירת מכונה זמינה. בנוסף, בטבלה זו מוגדרות המכונות הפעילות (active). ראוי לציין כי בעליית השרת, נטענות המכונות מחדש מקובץ הקונפיגורציה, כדי לאפשר הוספת והסרת מכונות, ולכן ערכי הפעילות מתאפסים. השרת אמור להישאר פעיל בכל עת ולכן מקרה זה צפוי לקרות רק במקרים של שינוי קונפיגורציה.
3. Requests – טבלה שכל כולה מוקדש לצורכי תקשורת. שירות ה-HTTP רושם בקשות עבור השרת המרכזי. השרת דוגם את הטבלה בהפרשי זמן קבועים (20 שניות, נכון להיום) ומוציא אותן לפועל. ממשק המשתמש מציג את הבקשות ומצבן.

## בנייה, אריזה וניהול גרסאות

לאורך פיתוח הפרויקט בצוות עשינו שימוש במערכת ניהול הגרסאות (Version Control System) Git ובכלי הבניה (Build tool) Apache Maven לצורך ניהול תלויות (Dependency Management). לאחר שלבי הפיתוח הראשוניים, כאשר נוצר הצורך לבנות ולארוז רכיבים לכדי קבצים הניתנים להרצה חיצונית לסביבת הפיתוח, עשינו שימוש נוסף ב-Maven: השתמשנו בפלאגין ליצירת JARים להרצת השרת והמכונות, בפלאגין אחר לאריזה (Assembly) של קבצי ההרצה עם כלל המשאבים הנחוצים (קבצי קונפיגורציה וקבצים חיצוניים אחרים) ובפלאגין נוסף לאריזה ייעודית עבור Servlet Containers אשר יצר קובץ WAR מוכן להצבה ב-Tomcat. יש לציין כי קובץ ה-WAR ניתן להצבה כמעט מיידית גם ב-Servlet Containers אחרים, דוגמת Jetty, אולם שינוי זה דורש התאמת קובץ קונפיגורציה (כלומר, כלל לא דורש שינוי בקוד עצמו).

# סיכום, מסקנות והצעות להמשך

במהלך השנה האחרונה שקדנו על פיתוח מערכת לניהול ולהרצת אנליזות בתחום חקר המוח. המערכת נבדקה על-ידי הרצת אנליזות סטנדרטיות על מידע גולמי מסוג EEG, לפי הנחיות מנחה הפרויקט. ריצת המערכת הניבה פלטים התואמים לפלטי האנליזות כפי שהיו מורצות באופן ידני טרם הפרויקט, ואיפשרה שיגור האנליזה, מעקב אחרי ריצתה וניהולה באמצעות ממשק המשתמש. כמו כן, נבדקו אפשרויות מתקדמות כמו ביצוע פעולות על משימות ו-Flows (ביטול, עצירה, הפעלה מחדש), שינוי המכונות הזמינות לשימוש המערכת בלחיצת כפתור, ושמירת אנליזות לשימוש חוזר.

המערכת שפיתחנו מכילה מספר רכיבים נפרדים, המתקשרים ביניהם במספר אופנים, ומורכבת מקרוב ל-10,000 שורות קוד בכ-7 שפות מחשב שונות. עקב מגבלות הזמן של הפרויקט ומורכבות המערכת, לא עלה בידינו לאפשר למשתמשיה תהליך של בחינת המערכת כדי לקבל את חוות דעותיהם. לכן, במידה ויתוכנן המשך לפיתוח המערכת, יהיה נכון בעינינו לאפשר לחוקרים תקופת הסתגלות למערכת ובעקבותיה קבלת הערות וחוות דעת. טרם קבלת פידבק מצד המשתמשים, אנו ממליצים על מספר שיפורים למערכת בהמשך:

1. שיפור מנגנון התלויות בין משימות – כיום המערכת מאפשרת להריץ במקביל מספר משימות, אולם כל משימה בעלת מספר סדרתי מסוים תלויה בהשלמת כל המשימות בעלות מספר סדרתי נמוך יותר באותו Flow (כלומר, השלמת כל היוניטים הקודמים). במקרים מסוימים, ניתן להפעיל משימה גם במידה ולא כל המשימות בעלות מספר סדרתי נמוך יותר הושלמו. לדוגמה, כאשר רץ Flow עם שני יוניטים על מספר נבדקים שונים, ניתן למקבל את המשימות ברמת הנבדק ולא רק ברמת היוניט. יש לציין כי כבר כיום ניתן להריץ במקביל Flows שונים ללא כל תלות זה בזה, ולכן זהו שיפור בנוחות אך לא מונע הרצה יעילה כבר כיום.
2. ניהול משתמשים והרשאות – כיום המערכת מאפשרת לכל משתמש עם גישה לשירות ה-HTTP לבצע פעולות ולא שומרת היסטוריה עבור משתמש ספציפי. ניתן להוסיף למערכת שכבת authentication שתדרוש מהמשתמש להזדהות ולהיות מוכר במערכת עצמה. במקרה זה ראוי להוסיף גם מסך אדמיניסטרציה שיאפשר ניהול משתמשים והרשאות. שיפור זה יאפשר שמירת היסטוריה מותאמת אישית, סיווג אנליזות מועדפות, וכן אלמנט נוסף של אבטחה מפני שימוש של גורמים בלתי מורשים.
3. ניתוח וטיפול בתקלות –

* גישה ללוגים מתוך ממשק המשתמש. כיום השרת, המכונות ושירות ה-HTTP רושמים כולם קבצי לוג. אלו אינם אמורים לעניין את המשתמש הפשוט, אולם במקרה תקלה, התבוננות בלוגים אמורה לאפשר הבנת הבעיה ופתרונה, ולכן גישה נוחה אליהם עשויה לשפר את השליטה. עם זאת, אנו צופים כי בדיקת לוגים לא תהיה תכופה ולכן לא ראינו לנכון לפתח יכולת זאת, שלא נכללה בדרישות הפרויקט.
* אפשרות ביצוע ריסט למכונה מתוך ממשק המשתמש. כיום המשתמשים יכולים להוציא ולהחזיר מכונות מאקטיביות ובכך לשלוט במכונות שיקבלו משימות להרצה, אולם אם קיימת בעיה באחת המכונות, על המשתמש לעצור את המכונה ולהפעילה מחדש שלא דרך הממשק. יש לציין כי במקרה של תקלה ראוי לבדוק את קבצי הלוג טרם עצירה והפעלה מחדש של מכונה.

מקורות

1. Azevedo, Frederico AC, et al. "Equal Numbers of Neuronal and Nonneuronal Cells Make the Human Brain an Isometrically Scaled‐up Primate Brain." Journal of Comparative Neurology 513.5 (2009): 532-541.
2. Fielding, Roy Thomas. "Architectural styles and the design of network-based software architectures". Diss. University of California, Irvine, 2000.
3. Gamma, Erich, et al. "Design patterns: Abstraction and reuse of object-oriented design." European Conference on Object-Oriented Programming. Springer Berlin Heidelberg, 1993.
4. Git online documentation, <https://git-scm.com/doc>.
5. Lerner, Ari. "Ng-book: The Complete Book on AngularJS". Fullstack.io, 2013.
6. O’Brien, T., et al. "Maven: By example. An introduction to Apache Maven." (2010).
7. RabbitMQ online tutorials, <https://www.rabbitmq.com/getstarted.html>.

1. <http://www.brainvoyager.com>. [↑](#footnote-ref-1)
2. מטלב (Matlab) היא תוכנה מבית MathWorks המאפשרת חישובים מורכבים ומשמשת במחקר ובתעשייה בתחומי המדעים וההנדסה. [↑](#footnote-ref-2)
3. ר"ת של Secure File Transfer Protocol. [↑](#footnote-ref-3)
4. ר"ת של Secure Shell. [↑](#footnote-ref-4)
5. ר"ת של Representational State Transfer. [↑](#footnote-ref-5)
6. ר"ת של Java API for RESTful Web Services. [↑](#footnote-ref-6)
7. <https://www.getpostman.com>. [↑](#footnote-ref-7)
8. הכתובת היא <http://rack-hezi-01.cs.tau.ac.il:9999/ubongo/rest/api>, נכון לכתיבת שורות אלה. [↑](#footnote-ref-8)
9. ר"ת של Software as a Service. [↑](#footnote-ref-9)
10. כונתה בעבר ng-grid. <http://ui-grid.info>. [↑](#footnote-ref-10)