REM: Reasonable Experiments-Manager

REM, כשתמומש, תיהיה תוכנת **קוד פתוח** בעלת ממשק גרפי ע"ג הדפדפן לתכנון ניסויים במערכות תוכנה מורכבות, מעקב אחר התקדמות הניסוי, זיהוי תקלות ופענוח התוצאות.

ממשק המשתמש יאפשר שיפור ביעילות העבודה עבור סטודנטים המבצעים ניסויים בכך ש:

- יקצר את זמן בניית הניסוי ע"י כלים לבניית מספר גדול של ניסויים השונים בפרמטרים בודדים, תוך מזעור הצורך בבניית קוד ייעודי עבור כל ניסוי.
- יאפשר קבלת אינפורמציה מלאה על הפרמטרים של הניסוי ואבחנה קלה עבור השוני בהגדרות בין מספר רב של ניסויים.
- יאפשר הרצה חוזרת (זהה) של ניסוי, עבור בקרה או לאחר תיקון תקלה, גם זמן רב לאחר הניסוי ללא צורך במאמץ עבור מבצע הניסוי.
- יקל על חיפוש ומעקב ביומן האירועים של הניסוי ע"מ למצוא תקלות ולוודא נכונות.
- יאפשר יצירת גרפים מורכבים ע"י ממשק משתמש נוח וידידותי בזמן קצר מאוד (on-the-fly)

עד כה, ע"מ להשיג מטרות אלו, נאלץ מבצע הניסויים לבנות סקריפטים מורכבים היעודיים לכל ניסוי בנפרד ועבור כל אחת מהמטרות הנ"ל בנפרד, וכן לקרוא את הסקריפטים הנ"ל ע"מ להבין בדיעבד את משמעות הניסוי.
המערכת תאפשר אוטומציה של פעולות אלו ותחסוך זמן רב מזמנו של מבצע הנסויים.
בנוסף, מערכת שכזו תאפשר למנחים לדעת שהסטודנטים עומדים בסטנדרטים גבוהים בכל הנוגע לתכנון ומעקב אחר הניסויים. כמו כן, המערכת תאפשר למנחים לעזור לסטודנטים לנתח תוצאות ניסויים בצורה מורכבת בזמן אמת גם במהלך פגישות קצרות.

לשם כך, אנו מציעים **3 פרויקטים** אשר יבוצעו **בזוגות**. הפרויקטים מוצגים להלן לפי רמת הקושי שלהם:

1. Log: ניהול יומנים (עמוד 3)
2. Configuration: תכנון ושחזור ניסויים (עמוד 4)
3. Plot: עיבוד נתונים בצורה גרפית (עמוד 6)

דרישות מהסטודנטים

המערכת מתוכננת במודל של Client/Server:

- Client - Front End: ממשק המשתמש יהיה דף דינאמי ע"ג הדפדפן - למעשה הינו תוכנת JavaScript.
- Server - Back End: שרת מבוסס Python ייתמוך בקריאה/כתיבה/עיבוד של הנתונים

הסטודנטים יוכלו לבחור אם לעבוד ביחד על כל הפרויקט או לחלק את העבודה לפי Front/Back-End.
לשם כך, לפחות אחד מהסטודנטים בצוות צריך להכיר טכנולוגיות של בניית ממשקי משתמש מורכבים ב-Javascript והכרה של סיפריות ייעודיות למטרה זו, כגון: JQuery ואחרות.
הסטודנט השני צריך שליטה ב-Python והכרה של הסיפריות הקיימות כגון: matplotlib לצורך בניית הגרפים.

הפרויקט יתנהל בשיטה אג'ילית: הפגישות יהיו דו שבועיות (גמיש), ובכל פגישה תוצג הדגמה (demo) עבור מספר user-stories שתוכננו לשבוע זה. כמו כן, הסטודנטים יסבירו אילו user-stories יוצגו בפגישה הבאה.

הרשמה

הפרויקטים מוצעים כאחד מהאופציות הבאות:

- Course 236371 (project in Concurrent and Distributed Systems), group 20.
- Course 236366 (project in operating systems).
- Course 236503 (project in Advanced Programming), group 20.

סטודנטים המעוניינים לקחת את הפרויקט או לשמוע יותר פרטים, מוזמנים לפנות אל:

לירן פונרו: funaro@cs.technion.ac.il

Log: ניהול יומנים

תיאור הבעיה

כיום, כדי לאתר תקלה או כדי לאבחן ניסוי, אנו נדרשים לעבור על מספר קבצי יומן. היומנים השונים מפוזרים בתיקיות שונות בשרת מרוחק. כדי לבצע חיפוש ביומן יש צורך לעבור על הקבצים ולהישתמש במילות מפתח. גודל הקבצים ומבוזרותם מאלצים אותנו לנווט בסיפריות ב-console ולקרוא את הלוגים בצורה טקסטואלית חסרת מבנה. כתוצאה מכך, קשה לבצע חיפוש עם מספר מסננים מורכבים. כמו כן, כשהמידע נמצא הוא אינו קריא בצורה נוחה.

תיאור הפתרון

- **GUI**: ממשק גרפי להצגת אוסף היומנים. דרישה זו מכילה הרבה מאפיינים. חלקם הכרחיים וחלקם ייתנו בונוס אם ייתאפשר הזמן. להלן המאפיינים לפי סדר חשיבותם:
  - הצגת היומנים בצורת טבלה ולפי שדות (ממויין לפי זמן)
  - טעינה ההדרגתית - היומן אינו נשלח בשלמותו, אלא רק את מה שניתן להציג כרגע על המסך
  - טעינה מתמשכת - הצגת יומן בזמן הרצת הניסוי - גלילה אוטומטית כשנוספות הודעות
  - סינון לפי פילטרים מורכבים
    - מילות מפתח
    - מילות מפתח בשדה מסויים
    - שילוב של תנאי AND/OR (תלוי בזמן)
    - מילים דומות (בונוס)
    - רעיונות יצירתיים נוספים ייתקבלו בברכה (בונוס)
  - הצגה צבעונית להבחנה פשוטה בין סוגי הודעות (תלוי בזמן)
  - יכולת "לבטל הסתרה" של מספר הודעות לפני ואחרי הודעה שעברה סינון (בונוס)
- **LogParser**: מודול זה מאפשר קריאת LOG ממספר מקורות ולהוציא פלט מאוחד, מחולק לשדות וממויין לפי זמן.
  - המודול צריך לאפשר קריאה ממספר סוגי LOG לפי הגדרות הקלט
  - סוגי הלוג יוגדרו ע"י מחלקות עזר שידעו לתרגם מידע מ-LOG טקסטואלי למידע המחולק לפי שדות
  - המחלקה צריכה לקרוא את הקבצים באופן "עצל": לקרוא רק את החלקים הרלוונטים בקבצים באותו הרגע

Configuration: תכנון ושחזור ניסויים

תיאור הבעיה

בנייה ושחזור ניסויים

כיום ע"מ לבנות מספר רב של ניסויים על כמות גדולה של פרמטרים במערכות תוכנה מורכבות, אנו בונים סקריפטים מורכבים עבור כל ניסוי בנפרד. בניית הניסוי מורכבת מבחירת מספר קבצי קונפיגורציה קבועים מראש וכן מפרמטרים שמועברים לפונקציות ע"י הסקריפט. על פרמטרים אלו אין מעקב והם לא נשמרים בשום מקום.
כאשר יש צורך בניתוח תוצאות הניסוי, אין שום אינדיקציה לפרמטרים שהועברו לצורך הניסוי. לכן אין יכולת לדעת בדיעבד מה היה מטרת הניסוי ואין יכולת לשחזר את הניסוי אם, למשל, התגלתה תקלה במהלך הניסוי זמן רב לאחר שהניסוי בוצע.

עדכון ניסויים

הרבה מהניסויים שנבנים משתנים עם הזמן. דבר זה מאלץ את מבצע הניסוי לעדכן את סקריפט הניסוי לניסוח מעודכן או חדש לגמרי. כתוצאה מכך, אנו מאבדים את מנגנון הניסוי הקודם שבוצע. אמנם ניתן לחזור אחורה בבקרת תצורה, אך לא תמיד הניסוי שבוצע בכלל עודכן בבקרת תצורה.

ניסוי על פרמטר

רוב הניסויים מכילים פרמטרים דומים מאוד אך שונים במספר פרמטרים מצומצם. ישנם מקרים בהם אנו מריצים אוסף ניסויים זהים בכל פרמטר מלבד פרמטר בודד המשתנה על סקלה רחבה או על מכפלה קרטזית של פרמטרים ע"מ לבדוק את השפעתם על תוצאות הניסוי.
כאמור, בנייה זו נעשית ע"י סקריפט ייעודי. אך כאשר בוחנית את התוצאות, לפעמים הרבה מאוד זמן אחרי הניסוי, אין אינדיקציה לגבי מה הפרמטרים שזהים בכל הניסויים ומה הפרמטרים המשתנים בניסוי. כמו כן, אין אפשרות להוסיף בקלות פרמטרים בניסוי ולבצע רק את הניסויים החסרים עבור הפרמטרים הנוספים.

תיאור הפתרון

- **GUI**: ממשק גרפי לבניית קבצי קונפיגורציה שיגדירו ניסוי באופן מלא, כך שלא יהיה צורך בנתונים נוספים ע"מ להבין ולשחזר את הניסוי. דרישה זו מכילה הרבה מאפיינים. חלקם הכרחיים וחלקם ייתנו בונוס אם ייתאפשר הזמן. להלן המאפיינים לפי סדר חשיבותם:
  - הגדרת הניסוי:
    - משתני סביבה
    - גרסת תוכנת בבקרת תצורה (revision) + שינויים שאינם בבקרת תצורה (patch)
    - קבצי קונפיגורציה (בפורמט python-dictionary)
    - מודולים נוספים הצריכים להיטען לצורך הניסוי שאינם חלק מהתוכנה הנבדקת
    - מודול הפייטון שיריץ את הניסוי (יקבל כפרמטר רק את קבצי הקונפיגורציה)
  - שכפול: בניית ניסוי ע"ב ניסוי ישן
  - בניית אוסף ניסויים המשתנים במספר פרמטרים או במכפלה קרטזית של פרמטרים (תלוי בזמן)

- מעקב אחר אילו ניסויים מתוך אוסף ניסויים בוצעו, אילו הצליחו ואילו נכשלו (תלוי בזמן)
- עדכון פרמטרים לאוסף ניסויים כך שנוכל להשלים את הניסויים רק עבור הפרמטרים החדשים (תלוי בזמן)
- השלמה אוטומטית של פרמטרים ושמות מחלקות (בונוס)
- הצגה חזותית המקלה על קריאת הקונפיגורציה - למשל שימוש בצבעים והדגשות (בונוס)
- זיהוי ניסויים כפולים: סריקה של כל הניסויים וחיפוש כפילות בניסויים (בונוס)
- רעיונות יצירתיים נוספים ייתקבלו בברכה (בונוס)

- **Parser**: מודול פייטון שידע לקרוא את קבצי הקונפיגורציה ולטעון את המודולים הנוספים
  - מודול הניסוי (**לא בסקופ של הפרוייקט**) יקרא את הגדרות הניסוי בעזרת מודול זה.
- **Executer**: סקריפט שיריץ את הניסוי:
  - יגדיר משתני סביבה
  - יעביר את התוכנה לגרסה הנבחרת
  - יבצע רישום של פרמטרים שאינם בשליטת הניסוי כגון: תאריך, גרסת מערכת ההפעלה, סוג המכונה עליה רץ הניסוי, וכדומה
  - יריץ את פקודת הניסוי כאשר הפרמטר היחיד הינו ספריית קבצי הקונפיגורציה
    - קריאת הקבצים ע"י המודול הייעודי (Parser) הינה באחריות סקריפט הניסוי

Plot: עיבוד נתונים בצורה גרפית

תיאור הבעיה

לאחר ביצוע הניסויים, נאגר מידע רב מעשרות מוניטורים ומעשרות רכיבים שונים במהלך הניסוי. מבצע הניסוי לא תמיד יודע מה המידע שנאגר ולא יודע לבחור אילו גרפים לייצר ע"מ להסיק מסקנות מעניינות מהניסויים.
מכיוון שבהינתן, לדוגמה, מוניטור חדש, יצירת גרף מתאים דורשת מאמץ תכנותי, מבצע הניסויים נאלץ לבחור בקפידה על אילו גרפים הוא ישקיע את המאמץ הנדרש. בנוסף, אם בזמן פגישה, הגיעו למסקנה שיצירת גרף נוסף תעזור בהבנת תוצאות הניסוי, יצירת הגרף תאלץ להמתין לפגישה הבאה.

תיאור הפתרון

- **GUI**: ממשק גרפי המאפשר יצירת גרפים בפשטות. דרישה זו מכילה הרבה מאפיינים. חלקם הכרחיים וחלקם ייתנו בונוס אם ייתאפשר הזמן. להלן המאפיינים לפי סדר חשיבותם:
  - בחירה מאוסף מודלים מוכנים לייצירת גרפים המתאימים לנתונים בניסוי זה
  - יצירת גרפים בקליק והצגתם
  - יצירת מודל חדש לגרפים
    - בחירת מספר שדות להצגה בציר הזמן: בחירת מתוך רשימת כל השדות הנתונים מהפלט של הניסוי
    - בחירת שדה מפלג: עבור כל ערך שונה של השדה יווצר גרף אחר או קו חדש על אותו הגרף (למשל לפי שם התהליך הנעקב)
    - בחירת ערך להשוואה בשדה מפלג: בהינתן שדה מפלג, ערך מסויים בשדה זה יוצג בכל הגרפים כך שניתן יהיה להשוות אותו לערכים ביחס לשאר הערכים בשדה
    - בחירת אופי הנתונים:
      - דגימה מעידה על העתיד: כל הנתונים בין דגימה ראשונה עד לדגימה הבאה זהים לדגימה הראשונה
      - דגימה מעידה על העבר: כל הנתונים בין הדגימה הראשונה עד לדגימה הבאה זהים לדגימה הבאה
      - אינטרפולציה לינארית: הערכים בין כל שני דגימות הם קו ישר המחבר בניהם
    - בחירת ערך שונה במקום ציר הזמן: לכל ערך בשדה א' מה היה הערך שנדגם באותו הזמן בשדה ב' (תלוי בזמן)
      - בחירת התמודדות עם מספר דגימות עבור כל ערך:
        - ממוצע - עם מדדי שגיאה
        - מקסימום
        - מינימום
        - הצגת scatter של כל הערכים
      - בחירת ערך מפלג וערך להשוואה באופן דומה למקודם.
    - לכל מכפלה קרטזית של ערכים בקבוצה של שדות, מה היה הערך הנדגם באותו הזמן בשדה ג' (בונוס) - זהה למקודם אך בגרף רב מימדי
  - הכנסת plugins בפורמט של מודולים חיצוניים ב-Python ליצירת גרפים מורכבים יותר (בונוס)
- **Database**: מבנה נתונים בדיסק שיאפשר איחסון של המודלים השונים שהוכנו לשם שימוש חוזר.

- ה-DB צריך להיות קובץ בודד שקל להעביר ביחד עם התוכנה
- הוא צריך להיות יעיל מספיק לקריאה ושכתוב הנתונים ללא כתיבת כל הקובץ מחדש (XML לא עונה על דרישה זו)
- אין צורך בתמיכה במקביליות
- פורמט ה-DB צריך להיות מוכר ובשימוש נפוץ כך שייתאפשר לקרוא ולערוך את המידע בצורה ידנית במקרה הצורך
- רצוי וממולץ שלפורמט זה תהיה ספריה בשימוש נרחב עבור Python

- **Plotter**: בניית ה-DATA שממנו ייבנה הגרף.
  - פלט המודול (DATA) צריך להכיל את כל הנתונים כדי ליצור את הגרף כך שלא יהיה צורך בקריאת נתונים נוספים מפלט הניסוי.
  - ה-DATA צריך להכיל את הפרמטרים שניתנו למודול כדי ליצור את ה-DATA הזה בהינתן הנתונים מהניסוי.
  - המודול צריך לאפשר הזנת הפרמטרים באופן "ידני" בקוד כדי ליצור את אותו המידע בדיוק.
- **FigureMaker**: מודול היוצר את הגרף בהינתן ה-DATA שנוצר מה-Plotter.
  - המודול צריך להיות מסוגל ליצור את הגרף בהינתן ה-DATA בלבד ללא קריאת נתוני הניסוי עצמו וללא פרמטרים נוספים