

**נושא העבודה: CalcFarm**

**התמחות: הגנת סייבר**

**לוגו :**

תמונה שמכילה לוח ציור, טקסט

התיאור נוצר באופן אוטומטי

**שם התלמיד: איתי ברק**

**ת.ז.התלמיד: 322879545**

**שם בית ספר ועיר: אורט שפירא, כפר סבא**

**שמות המנחים והמלווים: צביקה שטרקמן ומוטי גורנשטיין**

**מועד הגשת הפרויקט: 29/5/20**

**תוכן עיניינים**

* מבוא – ע"מ 3
* מדריך למשתמש – ע"מ 6
* הארכיטקטורה – ע"מ 14
  + פרוטוקול תקשורת – ע"מ 16
  + המודולים בפרויקט -
  + אלגוריתמים -
* בסיס הנתונים – ע"מ 26
* מדריך למפתח(יהיה בגרסה הסופית) – ע"מ 26
* סיכום/רפלקציה – ע"מ 27
* ביבליוגרפיה – ע"מ 32

מבוא:

בספר זה אני אתאר בפרטי פרטים על התוכן והטכנולוגיה מאחורי הפרויקט שלי Calcfarm.

Calcfarm הוא כלי מחשוב המאפשר לביצוע תהליך חישובי עצום של חישובים מתמטיים על תחום ענקי של מספרים על ידי חלוקת חישוב המטלה בין מחשבים שונים.

ישנה בעיה כאשר מחשב נשאל לבצע מטלת חישוב גדולה: יכול לקחת לו זמן רב לסיים, הוא יכול באמצע התהליך לקפוא, לא להגיב, אפילו לקרוס.

הפרויקט שלי פותר את הבעיה הזו על ידי חלוקת המטלת חישוב ליחידות קטנות ופיזורם בין כמה מחשבים שונים. ישנו שרת ראשי האחראי על המטלות של כולם, שרת עבודה המנהל את תהליך החישוב של מטלה, ופועלים, שהם עושים את העבודה השחורה ומחשבים את היחידות הקטנות מהמטלה.

התוכנה הזו מיועדת למשתמשים שיודעים לתכנת ובעיקר למתמטיקאים שרוצים לחקור השערות על תחום ענקי של מספרים.

עם התוכנה הזו, הם לא צריכים להשקיע בלקנות שרת מחשבים יקר ויכולים להשתמש במחשבים "רגילים" אחרים ברשת.

רקע

הרעיון לפרוייקט בא משתי מקומות: העניין וסקרנות שלי לגבי הכוח של חישוב מקבילי להריץ תוכנה גדולה בזמן קצר ומהעניין שלי במתמטיקה ובתורת המספרים. בזמני חופשי אני חוקר בעיות מפורסמות כמו השערת גולדבאך ואני משתמש בכלים רבים, ביניהם תוכנות שאני תיכנתתי שעוזרים לי לחקור את הבעיות הללו.

רציתי לשלב בפרוייקט את תשוקתי למתמטיקה ותכנות וליצור כלי למתמטיקאים שייעל את עבודתם.

תהליך המחקר

חלק משמעותי מהמחקר שלי היה מחקר עצמי על מחשוב מקביל:

בהתחלה של הפרויקט חשבתי וישבתי עם המנחים שלי לסייבר לחשוב מה הדרך הכי יעילה לחלק משימה מחשבתית בין כמה מחשבים, ככה שהתוצאה תהיה הכי יעילה, מהירה ודינאמית.

בהתחלה אני חשבתי פשוט לחלק את התחום מספרים בנאלית, פשוט שכל מחשב מתוך n מחשבים המחוברים, יקבל מהמספרים. אם התחום מספרים הוא בין 1 ל-100 וישנם שתי מחשבים מחוברים, אז מחשב אחד היה עובד על התחום מ-1 עד 50 והמחשב השני על התחום מ-51 עד 100.

בפרוטוטיפ הראשון של הפרויקט אני חילקתי מטלה בין שתי מחשבים אשר יעבדו בו זמנית על כמות שווה של מספרים שישלח אותם בחזרה לשרת.

אך בשלב מאוחר יותר, עם העזרה של המנחים, קלטתי שיהיה הרבה יותר יעיל לחלק את התחום מספרים לתחומים בעל גודל קבוע הנקראים "יחידות עבודה", שמחשבים יעבדו עליהם. ככה, אם מחשב קרס, אז במקום שאיבדנו את התוצאות של מהמספרים ואצטרך להוסיף אותם לתחומים של כל השאר, רק איבדנו תוצאות של יחידת העבודה שהשרת ייתן למחשב אחר. יחידות העבודה מאפשרות דינמיות בין המחשבים ושכל מחשב ינצל את מיטב משאבים למטלה: מחשבים חזקים יותר יעבדו קשה יותר וייקחו יותר יחידות עבודה ממחשבים חלשים יותר.

בנוסף לכך, אם התחום מספרים הכללי גדול מדי, אז נתקלנו באותה בעיה כי התחום מספרים של כל מחשב יהיה עדיין עצום.

המחקר העצמי הזה היה קשה, בהתחלה היו לי הרבה נעלמים שהייתי צריך לחשוב עליהם(יותר על זה בסיכום וב-"אתגרים") אך אני שמח שחשבתי ומצאתי את הפתרון שלי במקום להתבסס על מאמר על מחשוב מקביל עם הפתרון ה"נכון" וה"מקובל".

חלק שני מתהליך המחקר שלי, חקרתי קונספטים מתקדמים בתכנות כמו שימוש ב database וthreading שעוד לא נתקלתי בהם. בהתחלה לא הבנתי אפילו איך ליצור database בתוכנת הויזואליה.

בנוסף לכך, ניסיתי למצוא את החוקים הכלליים בשפה של sql כדי שאוכל לבנות ממנה מודול אשר מפשט ומכליל את sqlite(יותר מידע בחלק על מודולים בפרק על הארכיטקטורה).

בנוסף לכך, כמובן גם חקרתי את HTML, Javascript, ומודולים בפייתון שלא השתמשתי בהם עוד כמו bottle.

גם למדתי יותר לעומק את פייתון: מה הוא תכנות יפה ופייתוני.

אני למדתי ועשיתי את רוב המחקר שלי דרך האתר <https://www.w3schools.com/>, סרטונים על פייתון בyoutube והאתר מקור של bottle: <https://bottlepy.org/docs/dev/tutorial.html>

(כמובן גם עלי להודות למשיח Stack overflow שהושיע אותי מכל צרה).

אתגרים

במהלך המחקר המקדים ותיכנון הפרויקט נתקלתי בכמה אתגרים לגבי איך שהפרויקט יעבוד:

* אתגר 1: איך אני אדע אם תוכנת הפועל קפאה, קרסה או לא מגיבה?

קיבלתי הצעה להאריך כמה זמן אמור לקחת לכל מחשב לחשב יחידת עבודה ואם לוקח לו יותר מהזמן הזה, אז לנתק, אך יכול להיות שפשוט לוקח לו יותר זמן לחשב ואני מנתק אותו סתם.

* פתרון לאתגר 1: כל פועל ישלח כל תקופת זמן קבועה הודעת keep alive, ואם השרת מזהה שהזמן האחרון ששלח אותה היה מעל תקופת הזמן הזו, משמע שהפועל לא מגיב וצריך לנתק אותו
* אתגר 2: אחד מהאתגרים הכי קריטיים שנתקלתי בהם בתהליך התכנות המאוחר היה שהזמן שלקח לתוכנה לבצע מטלה, אפילו עם כמה עובדים, לא נקטן, אלא נשאר קבוע.
* פתרון לאתגר 2: הגעתי למסקנה שכמות המספרים ביחידת העבודה הייתה קטנה מדי. למחשבים המודרניים של היום לוקח מילישניות לעבור על אלפי מספרים, אז כאשר הרצתי את הפועל, לתקשורת לקח יותר זמן מאשר חישוב המטלה, , כלומר לתקשורת היתה תקורה (overhead) שהגדילה את הזמן לביצוע תת המטלה. אני פתרתי את זה בכך שהגדלתי את כמות המספרים שיש בכל יחידת עבודה, ככה שלקח למחשב חזק כמו שלי לפחות כמה שניות. כעת, משום שלוקח זמן, הפועלים חישבו יחידות עבודה בו זמנית, בעוד קודם לקח מילישנייה לחשב, לכן כמעט אף פעם לא יצא שהם חישבו בו זמנית ועזרו זה לזה.
* אתגר 3: אתגר שחזר בצורות שונות הוא שאני עובד עם שרתי bottle סטטיים שאינם יכולים לרוץ מעצמם, רק שלקוח(פועל או שרת אחר) ניגש לאחד מה-routes שלהם: הם לא יכולים לשלוח הודעות לפועלים להתחיל או להפסיק לעבוד, הוא לא יכול אפילו לבדוק בעצמו כל תקופת זמן מתי הפעם האחרונה שהפועל
* פתרון לאתגר 3: מצאתי שתי פתרונות לאתגר: הקל מביניהם, שירוץ עם השרתים thread נפרד שהוא מריץ דברים כמו בדיקת מצב הפועלים ודרך הפרוטוקול תקשורת שפיתחתי(הסבר עליו יותר מאוחר)
* אתגר 4: אחד מהאתגרים הכי קשים שהתמודדתי איתם הוא שהיחידות לא הצליחו לתקשר זה עם זה: לקראת סיום הפרויקט, השרת הראשי לא הגיב לשרת העבודה ואחרי שניסה כמה פעמים, הפסיק להריץ מtimeout.

שברתי את הראש על הבעיה הזו שעות רבות, בוזבזו הרבה ימים רק על כך שניסיתי לפתור את הבעיה, אפילו המדריך שלי, מוטי, לא הצליח לפתור את הבעיה

* פתרון לאתגר 4: גילית ממוטי, ששרתי bottle אינם multithreaded כמו כל שרת קיים, אלא הוא רץ רק על thread אחד: משמע שכאשר כמה לקוחות ניגשים לשרת בו זמנית, השרת מטפל בבקשות אחד אחר אחד.

זה הסיבה למה לקח לשרת העבודה הרבה זמן להגיב ליחידות השונות. אז החל

לא משנה כמה אתגרים היו בפרוייקט הזה, תמיד היה לי מוטיבציה כי יש לי תמיד תשוקה כאשר אני יוצר משהו שלי שהוא מקורי, בין אם זה אומנות או תכנות וכאשר אני רואה שזה עובד כפי שתכננתי, אני מתמלא עם סיפוק, מוטיבציה ואושר שאני יוצר כלי מחשוב עוצמתי חזק שיכול לעזור לאחרים.

מצב בשוק

טכנולוגיה כזו של עיבוד מקבילי אכן קיימת כבר בשוק: ישנם מתמטיקאים המשתמשים בכוח של סופר מחשב המורכב מעשרות מחשבים כדי לפתור בעיות, לgoogle יש שדה של שרתים המטפלים במטלה הענקית של חיפוש בכל חלקי הרשת העולמית המתבצע כמה פעמים בשנייה משום שבאים אלפי בקשות בשנייה מכל העולם.

הדבר הכי קרוב למוצר שלי הוא שירות הנקרא Charity engine, אשר מאפשר לבצע תוכנה ברקע של 500 מחשבים[. היה אפילו מתמטיקאי שמצא סכום של שלושה מספרים בחזקת 3 ל42 דרך שירות זה.](https://www.eurekalert.org/pub_releases/2019-09/uob-sot090619.php)

למרות זאת התוכנה שלי מאפשרת לכל אחד להשיג כוח חישובי עצום. ממש כל אחד, גם אם יש לו כמה מחשבים בבית, או כמה מחשבים בית הספר או במעבדת מחקר פשוטה: לא צריך לקנות ציוד מורכב ויקר כמו ליבות נוספות או מחשבים קטנים (או ללמוד docker כפי שקראתי במדריך) אלא רק צריך להוריד את התוכנה ולהפעיל אותה. אפילו אם למישהו יש רק מחשב אחד אם 3 ליבות, אז כבר התוכנה תרוץ פי 3 יותר מהר ואם למזלו 3 מחשבים, אם יש בכל מחשב 4 ליבות, אז יהיה לו פי 12 יותר כוח עבודה פוטנציאלית מאשר אם הריץ את התוכנה כרגיל.

**מדריך למשתמש:**

שלום משתמשים יקר, אני שמח שבחרתם להשתמש במוצר שלי.

מה שיש לכם הוא כלי חישוב עוצמתי שמאפשר לכל אחד לרתום כמה כוח מחשוב שאתם רוצים לתהליך חישוב ענקי.

אם אתם מריצים על המחשב שלכם לולאה העוברת על כל המספרים מ-1 עד 10 בחזקת 18, יקח לו נצח, אם הוא לא יקפא או יקרוס. עם Calcfarm, אתם יכולים להריץ אותה מהר יותר ויעיל יותר.

זה נעשה בכך שהתוכנה מחלקת את מטלת החישוב בין מחשבים: ככול שיש לכם יותר מחשבים המחוברים לתוכנה, ככה גדל משמעותית כוח החישוב שלרשותכם.

התוכנה הזו נועדה למשתמשים שאוהבים תכנות ומתמטיקה, שאוהבים לשבור גבולות, ולחקור אמיתות.

דרישות

* מומלץ שהתוכנה תרוץ על windows
* שלמחשב יהיה קישור חזק לאינטרנט
* דרוש ידע לתכנת בפייתון ושיהיה מותקן פייתון על המחשב שלך כדי לתכנת, לערוך ולבדוק את קובץ המטלה שלך.
* דרוש שהתהליך חישוב שלכם יהיה לחקור תחום מספרים עצום שיהיה אפשר לחלק אותו לתחום קטנים יותר.
* דרוש להוריד את הפקטה של Calcfarm: שתי התוכנות: Work\_server.py וWorker.py עם כל המודולים הנחוצים להם על כל מחשב שאתם מתכוונים לרתום למטלה.

חוץ מהדרישות הללו, כמעט אין גבולות למה שהתוכנה יכולה לעשות. אם אתם יצירתיים, אתם יכולים אפילו להשתמש לתוכנה מעבר לתחום המתמטיקה, נגיד יש לכם בקובץ על מחשב מסוים מיליוני תמונות לגבות לcloud, אז תשייכו לכל תמונה מספר ושהמחשבים השונים יבקשו משרת נפרד על המחשב עם התמונות תמונה ויעלה אותם. כל יחידת עבודה המחשבים יעלו עשרת אלפים תמונות והתוצאה תהיה אם הוא בהצלחה העלה כל תמונה או לא.

מושגים ויחידות

* מטלה היא תוכנת חישוב שמיועדת להרצה על תחום מספרים עצום ולהחזיר פלט כלשהוא.

לדוגמא מציאת כל המספרים הראשוניים בין 1 ל5,000,000 או בדיקת השערת גולדבאך לכל המספרים בין 1 ל10\*10.

אם תחום המספרים הוא קטן יחסית: בין 1 ל1000 או אפילו בין 1 ל1,000,000, כבר יהיה עדיף להריץ את התוכנה כרגיל כי למחשבים המודרניים והמשוכללים של היום ייקח פחות מדקה, בעוד שלתוכנה שלי ייקח יותר זמן, מכל התקשורת וכתיבה במבנה נתונים. מתי שהפרוייקט שלי הכי יעיל כאשר תחום המספרים גדול כל כך שלוקח יותר מדי זמן או שהתוכנה נתקעת, נקפאת או קורסת.

המאפיינים של מטלה

* + המספר הראשון והאחרון בתחום המספרים הכללי של המטלה
  + האחוז כוח עבודה- האחוז הזה קובע את העדיפות של המטלה מהאחרות ואת אחוז כוח הפועלים שהיא דורשת. לדוגמא, אם האחוז שקבעתם הוא 50 אחוז, אז כמות הפועלים שיעבדו על המטלה הזו תיהיה בערך חצי מכמות הפועלים הכללית שרצים כעת ששלכם.

אם המטלה הזו דורשת יותר משאבים מאחרות(או מבחינת מורכבות הקובץ או גודל תחום המספרים), אז עדיף שהיא תקבל יותר פועלים שתסתיים מהר יותר.

* + קובץ פייתון- זהו לב המטלה ומה שמאפיין אותו, משום שזה מה שתוכנה תריץ על כל יחידת עבודה. כאשר כותבים אותה: הזמן שייקח לקובץ להריץ יחידת עבודה אחת תשפיע על כמות הזמן הכללית שייקח לפרויקט.

**אזהרה: תבדקו כמה שיותר את הקובץ הנ"ל שאין בו לולאות אין סופיות או רקורסיה שיכולה לגמור את התוכנה. תדמיינו שכל הפועלים מריצים את הקובץ הזה בו זמנית.**

* + ערך תנאי המטלה- יכול להיות שאתם לא רוצים שהתוכנה תעבור על כל תחום המספרים כי

אתם רוצים לבדוק השערה על מספרים ושהתוכנה תעצור כאשר נמצא מספר המקיים תנאי כלשהוא.

אז מטלה כזו נקראת "מטלה מותנת", וכל מטלה מותנת אוחז את "ערך תנאי המטלה" מסוים שיכול להיות כל טיפוס משתנה שפייתון תומך(שאינו None או מחרוזת ריקה) ויושווה לתוצאות של יחידות העבודה.

כאשר מטלה היא מותנת, צריך שהקובץ פייתון יחזיר tuple עם שתי איברים: באיבר הראשון תוצאה כלהיא, יכול להיות True או False לבין אם הוא קיים את השערה או לא או יכול להיות ערך מסוים שאתם מחפשים, ובאיבר השני יכול להיות מידע נוסף שיוצג אחר כך(כמו המספר הספציפי מתוך יחידת העבודה שקיים את התנאי). כאשר הערך הזה שווה ל"ערך תנאי המטלה, התנאי יתקיים והשרת יסיים לעבוד על המטלה

* יחידות העבודה: איך שהתוכנה שלי תריץ תחום מספרים שאפילו יפחיד מחשב מלהריץ הוא, כמו בחיים האמיתיים, לחלק מטלה ענקית למטלות קטנות יותר. התוכנה מחלקת את התחום הענקית לתחומים קטנים יותר בעל אותו כמות מספרים הנקראים יחידות עבודה.

בכל יחידת עבודה ישנה

ישויות הפרויקט

* השרת הראשי- הוא אחראי על כל המטלות של כולם: על חלוקת הפועלים בין המטלות ולתת לכל שרת עבודה מטלה לבצע. הוא רץ על השרתים של CalcFarm.
* שרת העבודה- הוא אחראי על תהליך חישוב מטלה.

הוא מחלק את המטלה ליחידות עבודה ודואג שכל יחידות העבודה תחושב ושולח את התוצאות לשרת הראשי.

כל שרת עבודה עובד על מטלה אחת ומטרתו היחידה היא שהמטלה הזו תבוצע ולשלוח את התוצאות לשרת הראשי, שיציג לכם. לכל מטלה שאתם רוצים לחשב, צריך שירוץ על אחד מהמחשבים שלכם שרת עבודה אחד.

* הפועל- התוכנה שאחראית על חישוב היחידות עבודה. התוכנה הזו מבקשת כל הזמן מהשרת עבודה יחידות עבודה, מריץ את קובץ הפייתון שאתם כתבתם עם התחום של יחידת העבודה שקיבל ואז הוא שולח את התוצאות חזרה לשרת העבודה.

התחברות

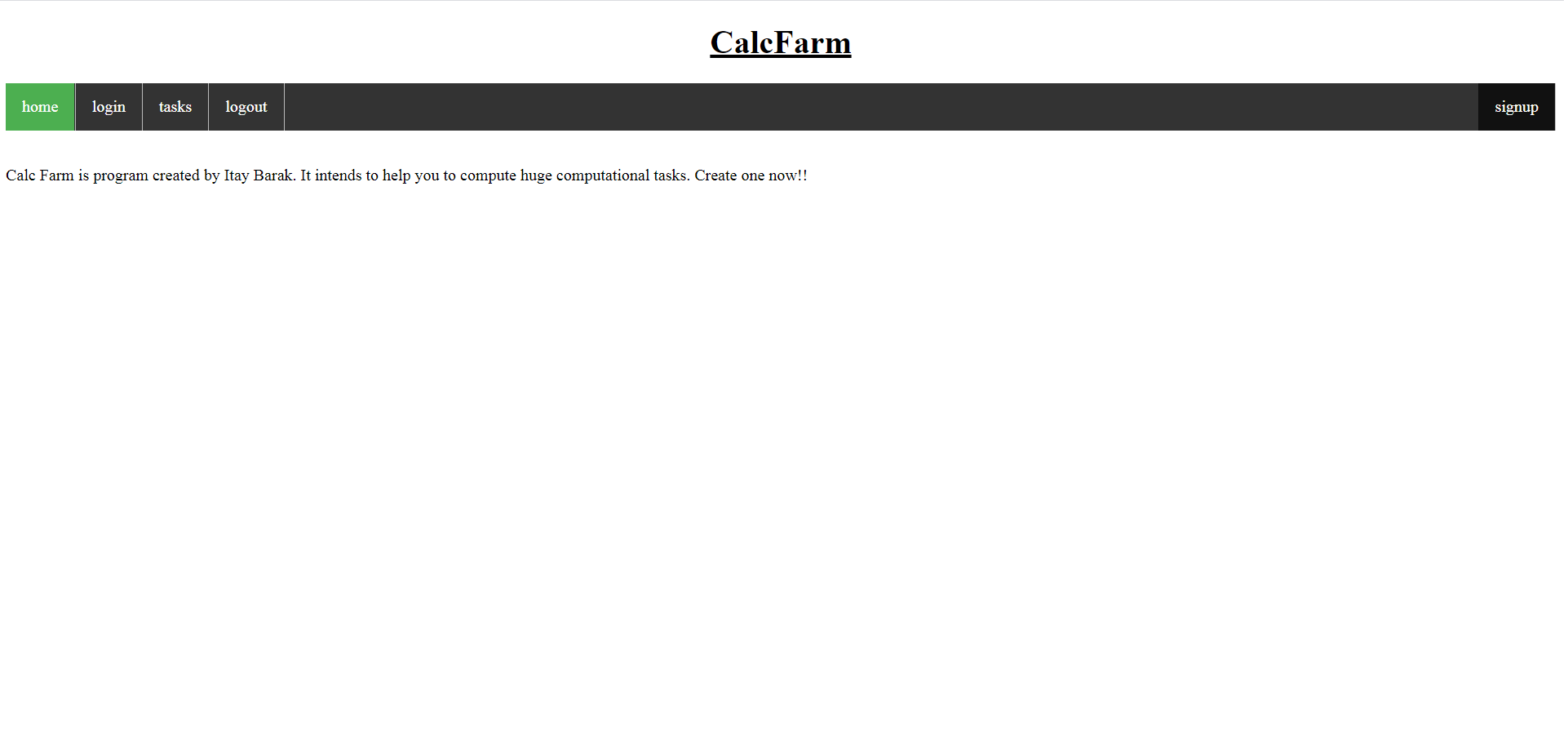
כל תהליך יצירת מטלות וניהול ביצועם מתבצע דרך האתר של CalcFarm:

ישנו בחלק העליון של כל עמוד באתר, מתחת לכותרת "CalcFarm", סרגל ניווט שממנו ניתן להגיע לכל עמוד באתר.

שתלחצו על הקישור, תגיעו לעמוד הבית (“home”) שכולל מידע כללי על הפרויקט.

יצירת משתמש(signing up)

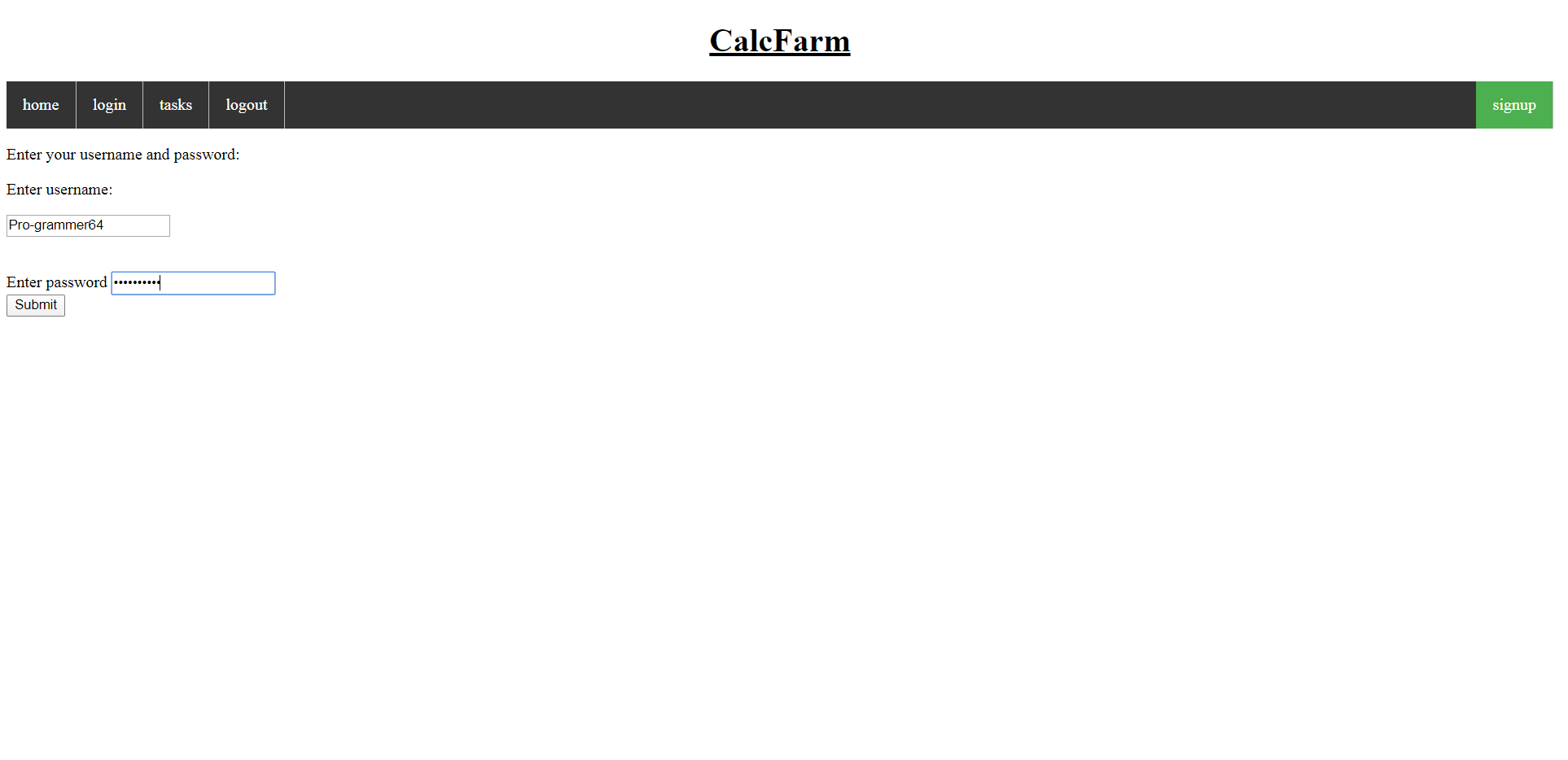
בשביל ליצור משתמש, גשו לעמוד "signup" הנמצא בצד הכי ימיני של הסרגל ניווט:



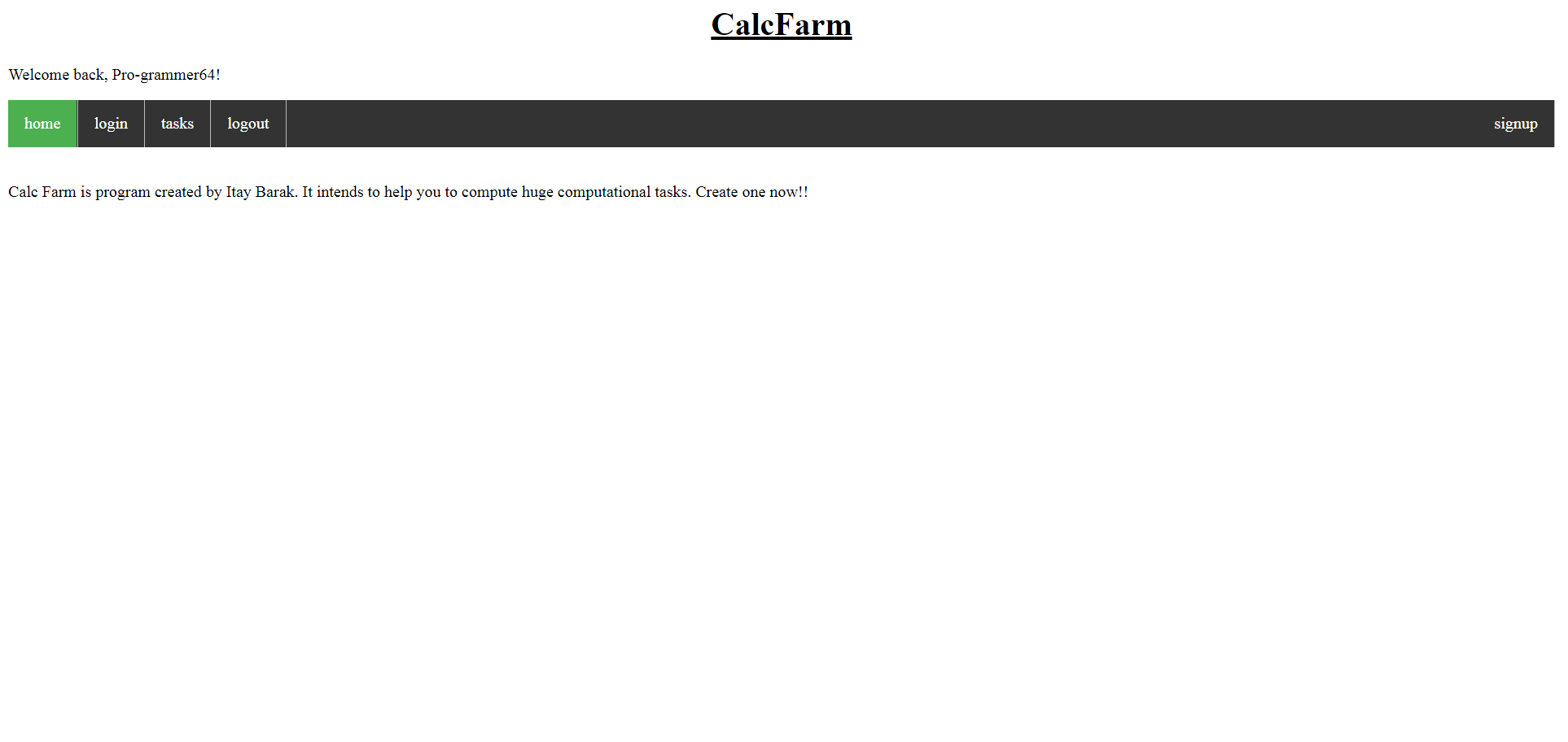


כתבו את שם המשתמש שלך בתיבת טקסט תחת "Enter Username"

וכתבו את הסיסמה שלך בתיבת טקסט תחת "Enter Password".



כאשר סיימתם למלא את הפרטים של החשבון שלך, לחצו על הכפתור " Submit" מתחת לשתי התיבות טקסט.



כעת, אחרי שלחצתם על ""Submit, אתם אוטומטית כבר מחוברים לאתר עם החשבון החדש שלכם(כדי לצאת מחשבון, קראו בהמשך).

התחברות למשתמש(logging in)

אם כבר יצרתם משתמש, כנסו לעמוד "login" שבסרגל הניווט, ושם כתבו את שם המשתמש שלך בתיבת טקסט תחת "Enter Username"

וכתבו את הסיסמה שלך בתיבת טקסט תחת "Enter Password".

כאשר סיימתם למלא את הפרטים של החשבון שלך, לחצו על הכפתור " Submit" מתחת לשתי התיבות טקסט.

תמונה שמכילה צילום מסך

התיאור נוצר באופן אוטומטי

כעת, אחרי שלחצתם על ""Submit, אתם אוטומטית כבר מחוברים לאתר עם החשבון שלכם(כדי לצאת מחשבון, קראו למטה).

יש בתחתית העמוד לינק לעמוד "signup" למטה למי שאין חשבון ורוצה ליצור(כתוב למעלה איך ליצור חשבון)

יציאה ממשתמש(logging out)

בשביל לצאת מהחשבון שלכם פשוט לחצו עם הכפתור “logout” שבסרגל ניווט:

ניהול המטלות

כל ענייני המטלות מתבצעות דרך העמוד "tasks"(אשר נכנסים אליו דרך סרגל הניווט למעלה).

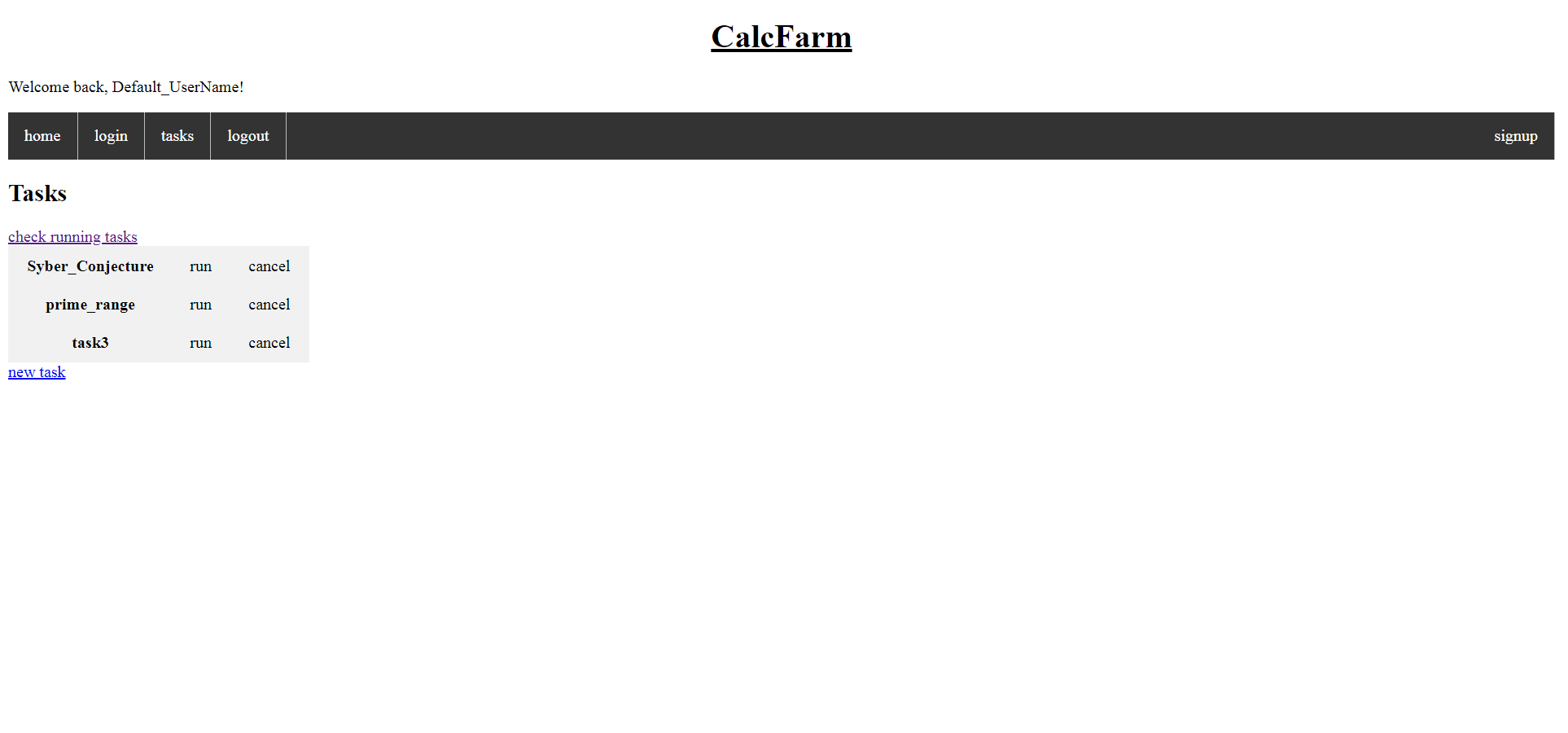
בשביל להכנס לעמוד זה אתם צריכים להיות כבר מחוברים לאתר עם החשבון שלכם. אם אינכם מחוברים ובכל זאת נכנסתם לעמוד, הוא יתן לכם לינק לעמוד הרשמה “log in”(אם אין לכם חשבון ובכל זאת נכנסתם לעמוד, הלינק השני מוביל ל"sign up"). ברגע שלחצתם על העמוד ונרשמתם(הוראות על איך לרשם נמצאות למעלה יותר במדריך), הוא יביא אתכם אוטומטית חזרה לעמוד “tasks” וייתן לכם הרשאה אל המטלות שלכם.

אם אין לכם חשבון עוד, לחצו על הלינק השני(הוראות על איך ליצור חשבון נמצאות למעלה יותר במדריך):

תמונה שמכילה צילום מסך

התיאור נוצר באופן אוטומטי

כאשר נרשמתם(או הייתם כבר רשומים לאתר), נפתח לכם הדף:



יש בו רשימה של כל המטלות שאתם יצרתם על החשבון שלכם, עם האפשרות לערוך או להריץ אותם.

יצירת מטלה

דרישות:

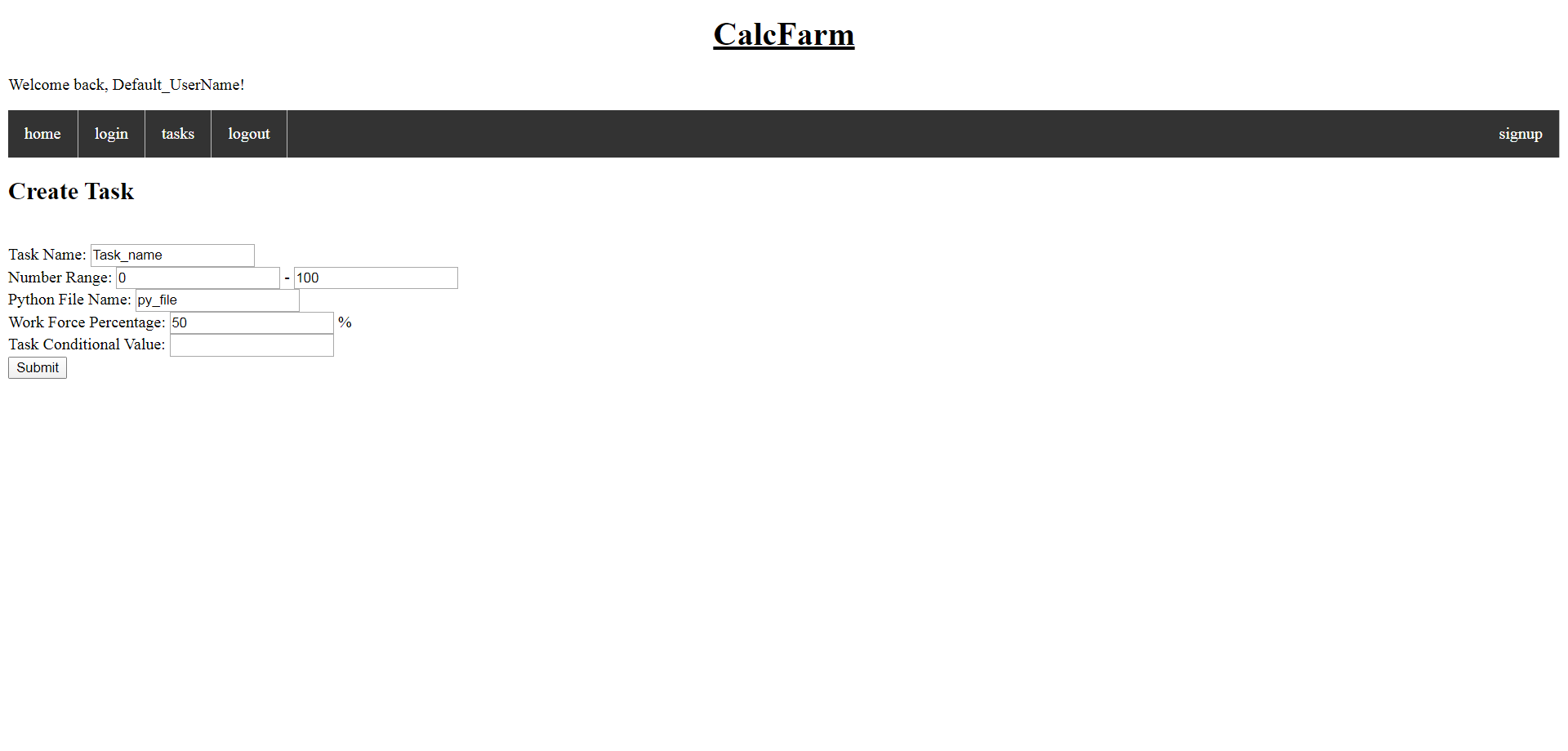
* קודם כל, כתבו את הקובץ של המטלה: הוא חייב להיות כתוב בקוד פייתון וחשוב שהוא יסתיים עם:

print(repr(results))

כאשר results שווה לפלט של הקובץ. משום שהתת תהליך מדפיס את הפלט שלו, אז הפועל רק יכול לקרוא מחזרות של התוצאה. אז כאשר התוצאות יודפס כrepr, והפועל יוכל להמיר את התוצאות חזרה ממחרוזת לסוג משתנה המקורי שלהם.

* התוצאות של הקובץ פייתון לא יכולות להיות גדולות מדי מבחינת כמות הביטים שלה, אחרת הפועל לא יוכל לשלוח את התוצאות לשרת כpost request

בשביל ליצור מטלה חדשה, עליכם להיכנס ללחוץ על הקישור בתחתית הדף new task"":



יוגש לפניכם פורום שבו ניתן למלא פרטים על המטלה:

* "Task Name"(חובה למלא)– השם של המטלה
* ""Number\_Range(חובה למלא)– תחום המספרים של המטלה: התיבת טקסט הראשונה עבור המספר הכי קטן בתחום והתיבת טקסט השנייה עבור המספר הגדול ביותר(התחום כולל את שני המספרים)
* "Python File Name"(חובה למלא)– השם של הקובץ פייתון שירוץ על כל יחידת עבודה(לכתוב רק את השם, בלי הסיומת של הpy בסוף)
* "Work Force Precentage"(אופציונלי) - האחוז כוח עבודה של המטלה. אם מכניסים לתיבה 50, האחוז יהיה 50%.
* "Task Conditional"(אופציונלי) – אם המטלה שלך אינה מותנת, תשאיר את התיבה הזו ריקה, אבל אם היא כן, לתיבה הזו אתה מכניס את הערך תנאי המטלה כrepr-. כדי שזה יעבוד עם כל סוגי המשתנים, אז אתה מכניס מחרוזת.

תיקח את הערך שנגיד נמצא במשתנה n, ואז מריץ בטרמינל פייתון או תוכנה: "repr(n)"

ואת התוצאה הזו אתה מכניס לשם(בלי המרכאות או הגרשים המקיפים את התוצאה).

לדוגמא אם repr(True) יצא 'True', אז כותבים רק True כי הגרשים רק כדי לסמן שהתוצאה היא מחרוזת. אך כאשר עושים repr למחרוזת, מתקבל שתי סוגי גרשיים: (=repr("answer""'answer'"

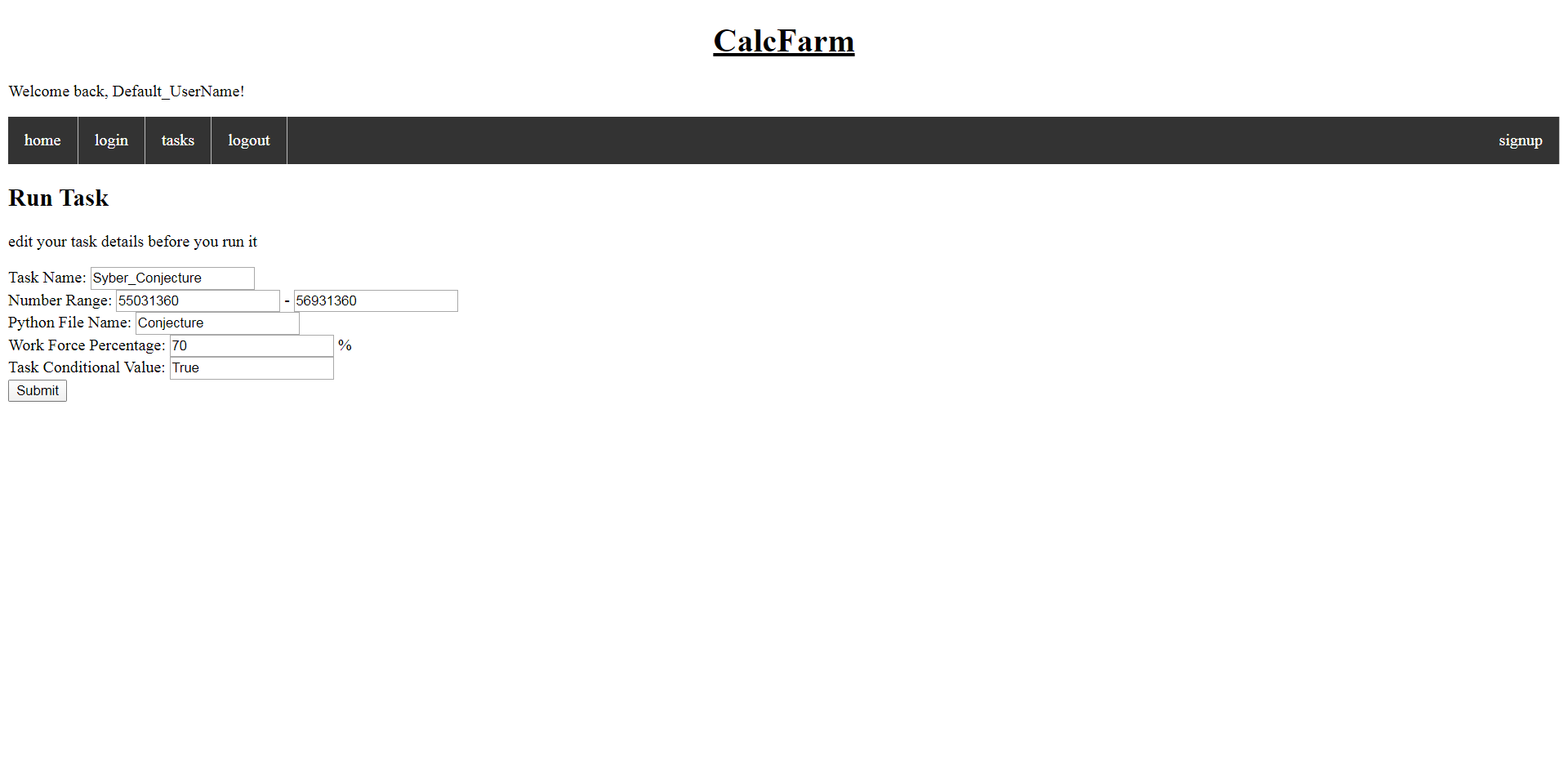
אתה רק נפטר מהמרכאות המקיפות את המחרוזת אך כולל את הגרשיים הפנימיים.

(מידע על כל אחד מהפרטים של מטלה בחלק ה"מושגים ויחידות" מוקדם יותר בפרק הזה)

לאחר מכן, לחצו על "submit".

הרצת מטלה

שתכנסו לעמוד "tasks", יהיה לפניכם רשימה של כל המטלות שיצרתם על החשבון שלכם. לחצו על הכפתור ליד המטלה שאתם רוצים להריץ שכתוב עליה "run"(אם אתם מתחרטים ורוצים להפסיק את הרצת המטלה, לחצו על הכפתור לידו "cancel").



האתר יאפשר לכם לערוך את פרטי המטלה לפני כל הרצה: בין אם אתם רוצים לבדוק תחום מספרים שונה או לשנות את האחוז. כאשר אתם לוחצים submit, זה ישנה את פרטי המטלה לצמיתות ויריץ את הגרסה החדשה של המטלה.

ניהול הרצת המטלה

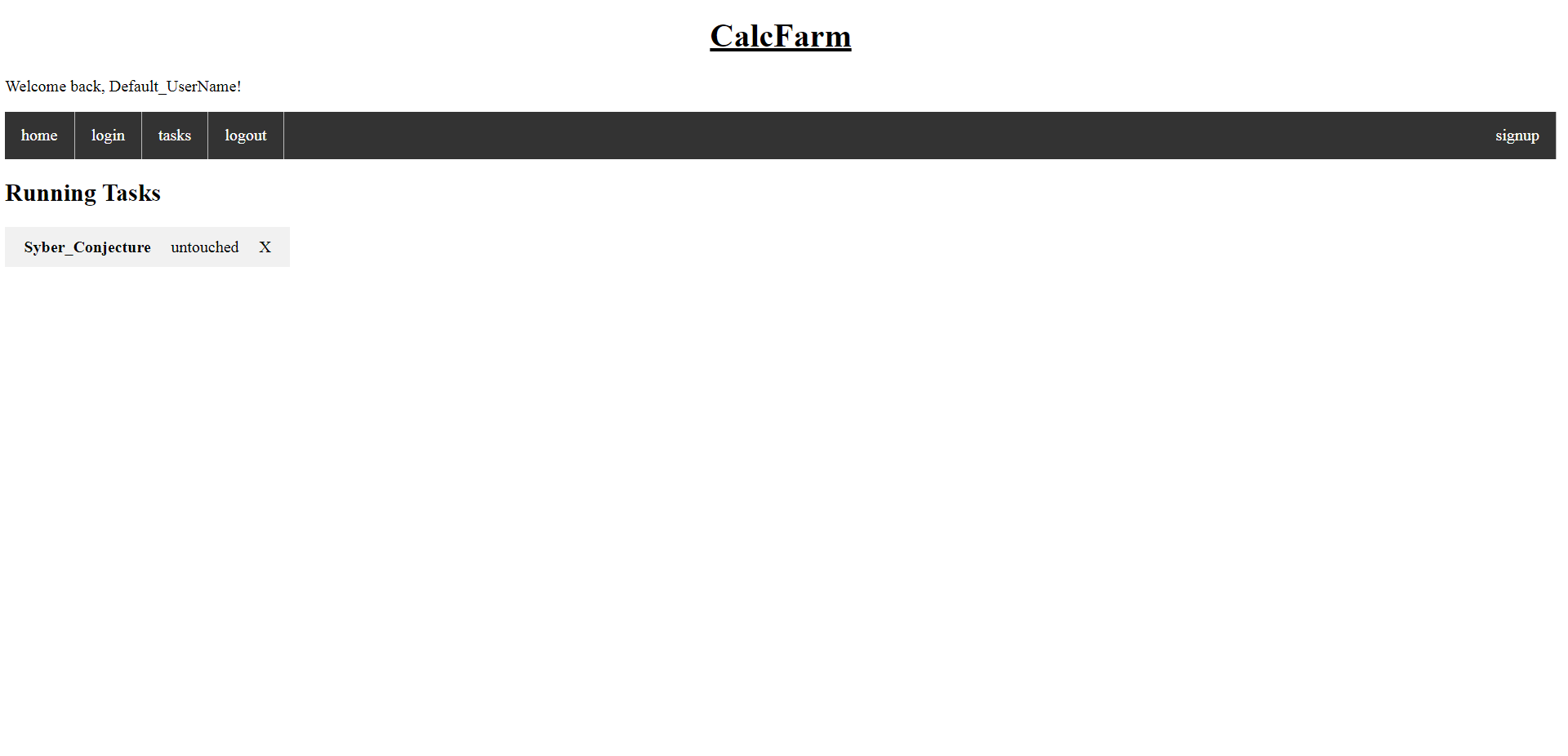
בעמוד “tasks”לחצו על הלינק שמעל הרשימת מטלות "check running tasks".

הוא יוביל אתכם לעמוד שבו תוכלו לנהל את כל המטלות שלכם שרצות כעת.

יש בה רשימה של כל המטלות ששייכות לחשבון שלכם שבתהליך חישוב.

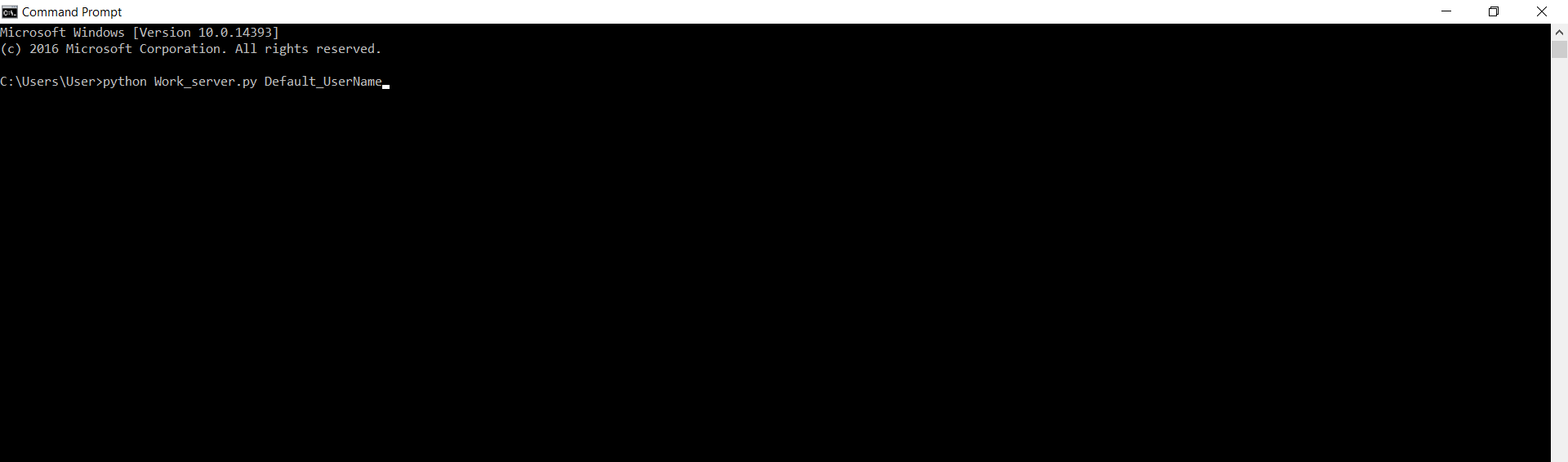
כמובן שאתם יכולים להתחרט גם שמה וללחוץ על הX שליד מטלה רצה כדי לבטל אותה.

המטלה מתחילה בסטטוס "untouched", משמע שאין שרת עבודה שעוד מויין לעבוד על המטלה הזאת. המטלה מחכה שאחד משרתי העבודה הפנויים שלכם ייקח את המטלה(אם אין שרתי עבודה שרצים ופנויים הוא יחכה עד שיהיה).

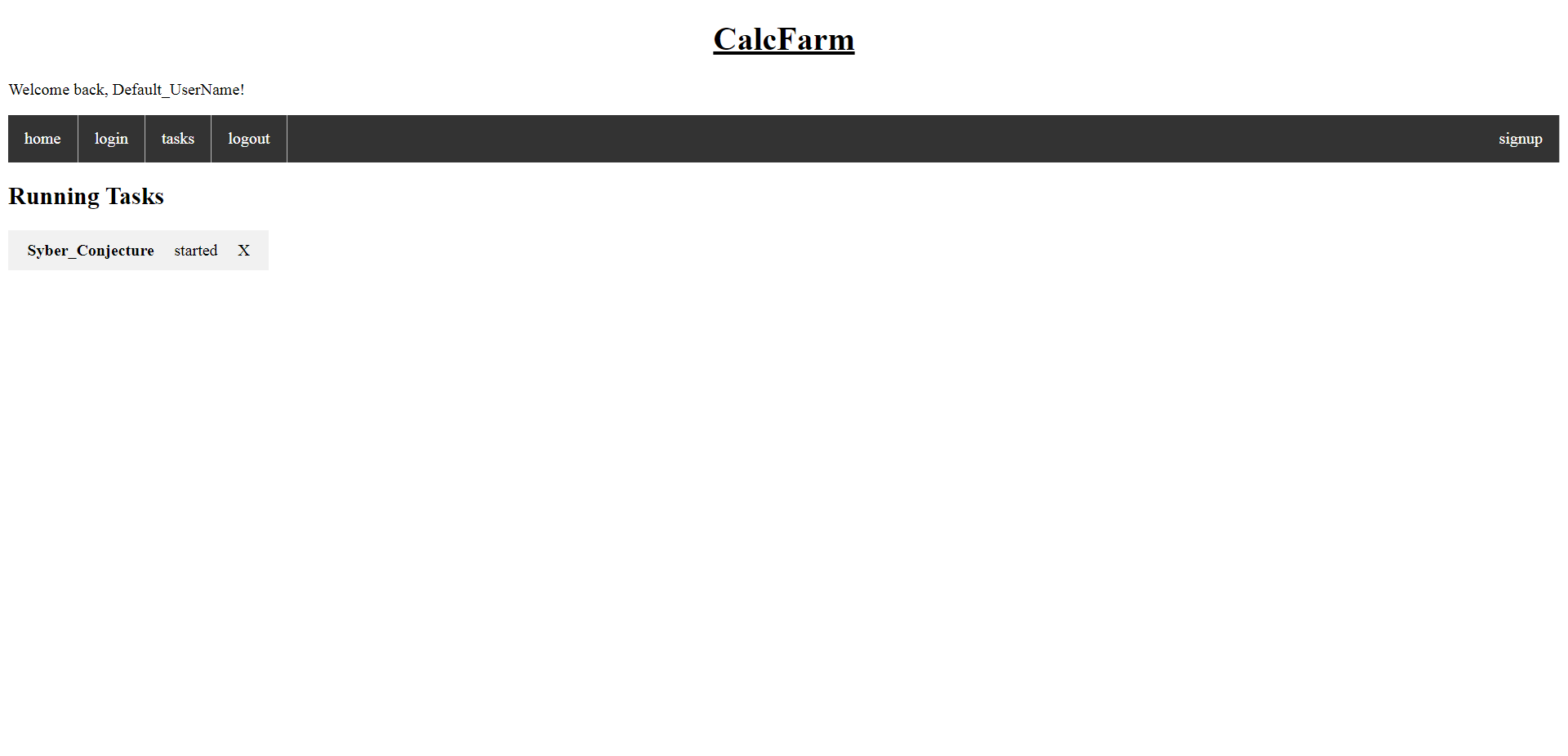


כדי להריץ שרת עבודה ופועלים שיעבדו על המטלות שלכם, בחרו מחשב מסוים ותריצו עליו את Work\_Server.py עם שם המשתמש שלכם כארגומנט:

ועל כמה מחשבים שאתם רוצים(יכול להיות על אותו מחשב שרץ עליו השרת עבודה), תריצו את Worker.py עם שם המשתמש שלכם כארגומנט.

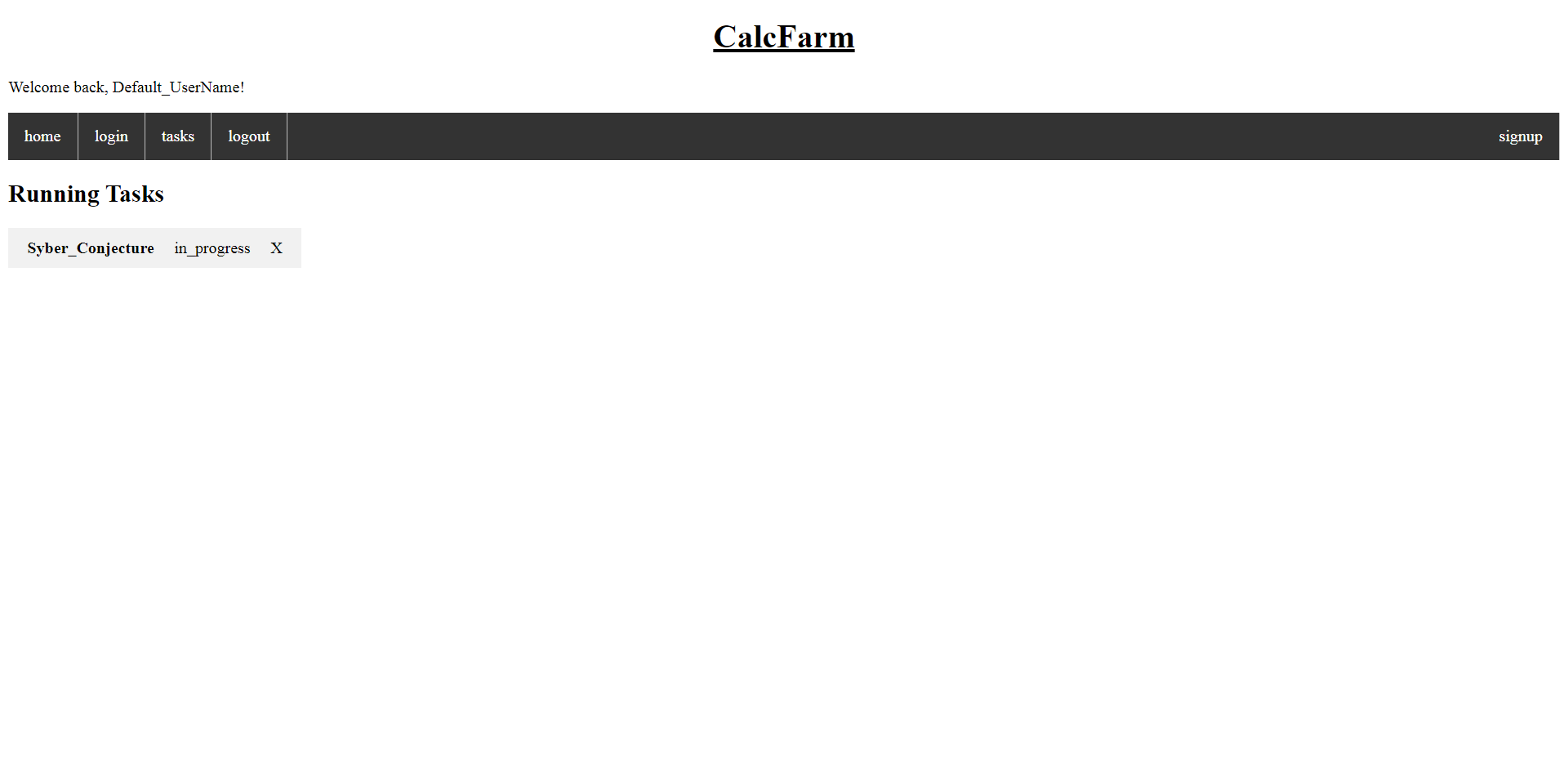


אם שרת עבודה לקח את המטלה, הסטטוס של המטלה ישתנה ל"started".

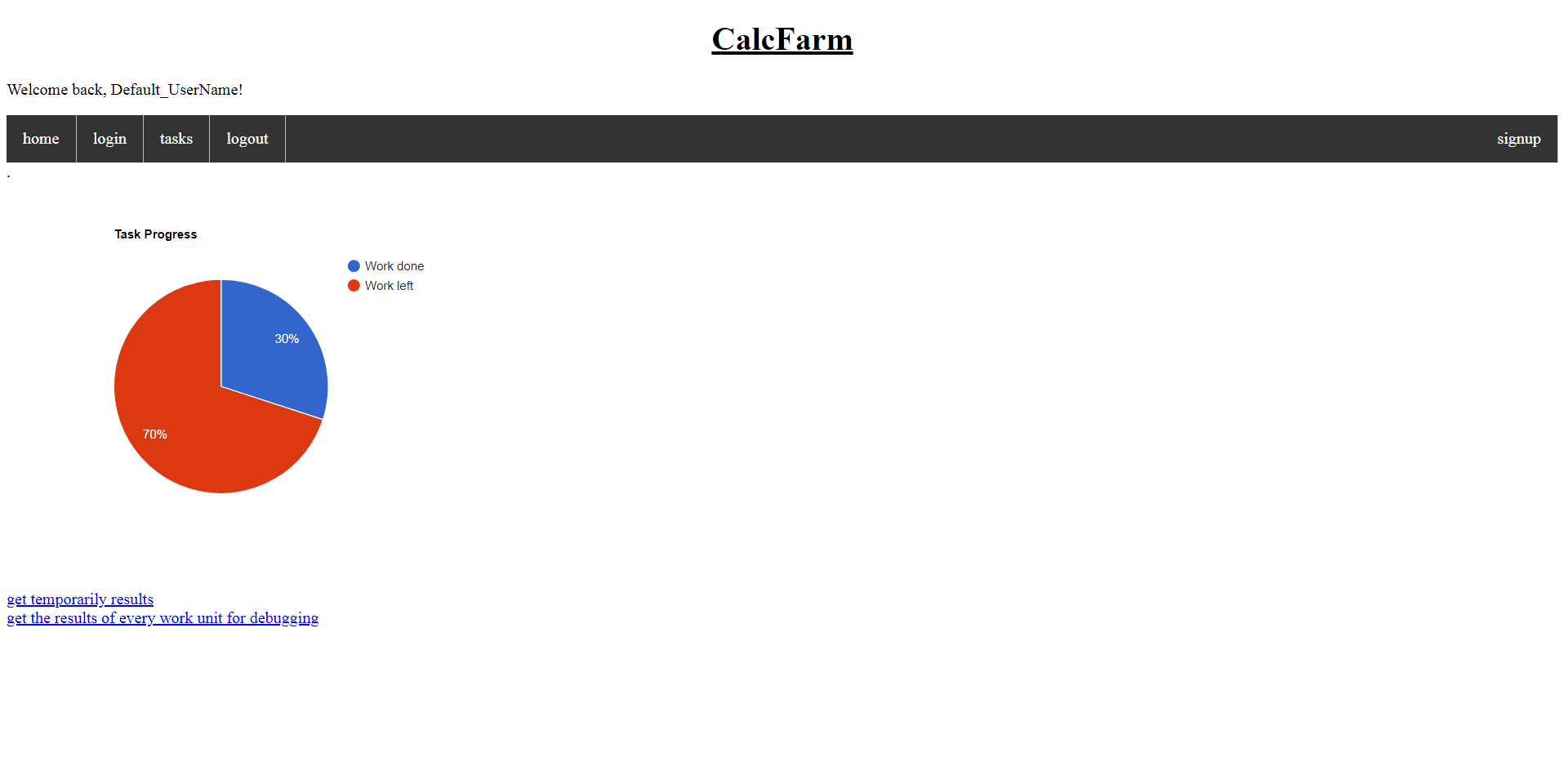


כעת השרת עבודה מחכה שפועל גם ימונה לעבוד על המטלה הזו ויתחיל לבקש ממנו יחידות עבודה(כדאי לדעת שהפועלים לא עובדים על מטלה ספציפית, אלא הם ייקשרו למטלה הזקוקה לפועלים. לדוגמא אם ישנה מטלה אחרת שרצה שקבעתם שזקוקה ל75 אחוז מהפועלים, יש סיכוי שהם יצטרפו למטלה הזאת מאשר למטלה החדשה).

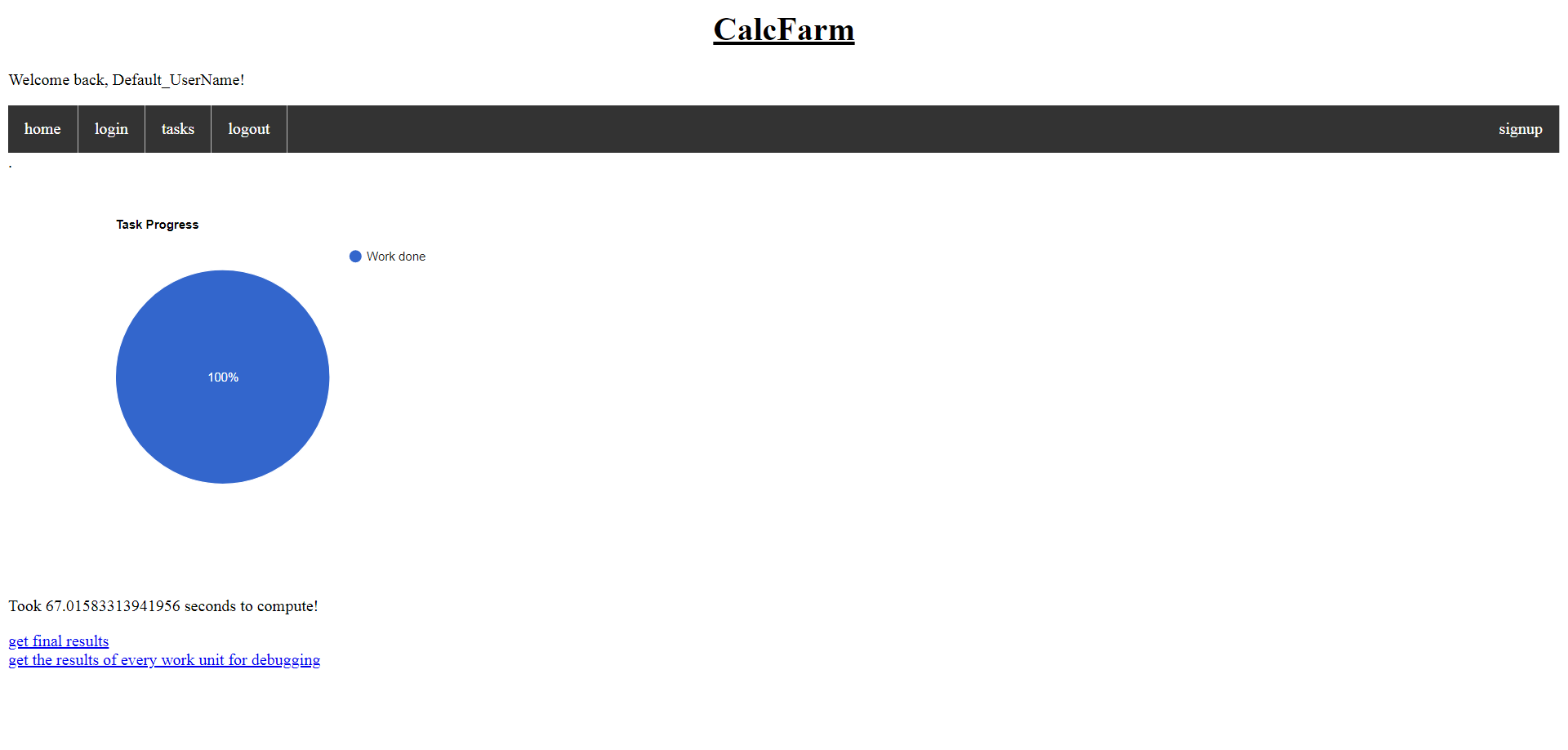
כאשר הפועל חישב את יחידת העבודה הראשונה, והשרת עבודה קיבל את התוצאות שלו, הסטטוס של המטלה ישתנה ל"in progress" והשרת עבודה ימדוד מהאירוע שקיבל את יחידת העבודה הראשונה עד שקיבל את האחרונה וסיים, כי עד כה השרת עבודה חיכה לפועלים ולא עבד.



כדי לראות את התוצאות של מטלה, תלחצו על שם המטלה ברשימה של מטלות רצות, ובכל רגע תראו גרף המראה את אחוז החישובים שבוצעו לעומת האחוז שנותר כדי לסיים את המטלה ולמטה יהיה לינק לשתי קבצים: בקובץ אחד ישנה מידע על כל יחידות העבודה עם התוצאות שלהם ובאחר כל התוצאות שנאגרו לרשימה אחת לפי סדר ID יחידות העבודה שלהם(התוצאות הראשונות ברשימה שייכות ליחידת העבודה הראשונה שנוצרה, ואז השנייה וכו...).



אם המטלה היא מותנת, אז בתוצאות יוצג רק התוצאות חישוב של יחידת העבודה שקיימו את התנאי: הtuple שאיברו הראשון הוא מה שהתוכנה השווה לערך התנאי והאיבר השני שבו ישנו מידע חשוב שאפשר לראות בקובץ.



**הארכיטקטורה של הפרוייקט**

תמונה שמכילה טקסט, מפה

התיאור נוצר באופן אוטומטי

הארכיטקטורה של הפרוייקט דומה לארכיטקטורה של חברת היי טק.

לדוגמא, בואו ניקח חברת היי טק שמייצרת כל מיני מוצרים.

יש את הבוס של החברה, הוא אחראי על ניהול העסק וכו...

ישנם כל מיני מטלות ופרויקטים שהוא מעוניין להרים בחברה(רעיונות חדשים למוצרים שהוא רוצה שיממשו, פתיחת סניף חדש, פתיחת שירות שונה לחברה) ותחתו ממומנים מנהלי משימה, שמטרתם היחידה היא לדאוג שהפרוייקט שהוטל עליהם יתממש במלוא. כל מנהל עבודה לוקח פרויקט וממנה קבוצה של עובדים חרוצים תחתו שמטרתם היחידה היא יחד לממש את הפרויקט על ידי חילוק העבודה ביניהם כך שכל אחד עובד על משהו אחר.

כאשר סיימו לעבוד על הפרוייקט הנוכחי, כל מנהל עבודה עובד על עוד פרויקט של הבוס(זה בסדר, למחשבים אין זכויות עובדים, הם יכולים לעבוד על כמה פרוייקטים שאני רוצה).

הבוס הוא השרת המרכזי, שרת עבודה הוא המנהל עבודה והפועל הוא תוכנת הפועל. ממש מבנה הירככי.

היחידות

הפרויקט מורכב מ3 תוכנות/ישויות מרכזיות המתקשרות זה עם זה:

שרת עבודה(Work\_Server):

* שרת העבודה הוא אחראי על ניהול תהליך חישוב המטלה
* הוא לוקח מהשרת המרכזי מטלה שהמשתמש המשוייך אליו(הוא ישוייך בכך שהקוד של השרת ירוץ עם הארגומנט של שם המשתמש) רוצה שייבוצע עם תחום המספרים הכללי של המטלה ואת הקובץ שירוץ על כל מספר בתחום מספרים הזה.
* הוא יחלק את התחום מספרים לתחומים קטנים בעל גודל קבוע של מספרים הנקראים "יחידות עבודה"
* הוא מתקשר עם הפועלים, שולח להם את הקובץ שהם צריכים ו"מאכיל" אותם עם יחידות עבודה לחשב.
* הוא אמור לקבל מהפועלים תוצאות של יחידות העבודה שהם לקחו ולשמור אותם בdatabase איפה ששמור יחידת העבודה
* הוא גם אמור לשמור על הסטטוס של כל הפועל ולבדוק אם הם עובדים או מגיבים.
  + הוא אמור בthread נפרד מהשרת לבדוק כל יחידת זמן את הזמן האחרון שפועל שלך לו מסר, ואם הוא יותר מזמן מסוים, הוא מנתק אותו, מוחק אותו מהDatabase ו"משחרר" את יחידת העבודה, שפועל אחר ייקח אותה.
  + אם הפועל שולח לשרת שקרס או שהתוכנית שלו הסתיימה, השרת ינתק אותו, ימחק אותו מהDatabase ו"ישחרר" את יחידת העבודה, שפועל אחר ייקח אותה.

פועל(Worker):

* הוא אמור לספק כוח חישוב לכל מטלה שהמשתמש המשוייך אליו (הוא ישוייך בכך שהקוד של הפועל ירוץ עם הארגומנט של שם המשתמש) רוצה להריץ.
* הוא ייקח יחידת עבודה מתוך המטלה ויריץ את הקובץ מטלה (שייקח מהשרת עבודה אם אין לו) על התחום החלקי שקיבל.
* הוא אמור לעדכן את השרת עם התוצאות שקיבל והמצב הנוכחי שלו(יותר מידע בפרוטוקול התקשורת)
* הוא אמור כל תקופת זמן קבועה לשלוח לשרת מסר שאומר שהוא עוד רץ ומחשב בthread נפרד.

שרת מרכזי(Main\_Server):

* יש לו שתי תפקידים:לתפעל את האתר ללקוחות ולתפעל את מטלות של המשתמשים
* הוא שומר נתונים על כל המשתמשים: השמות משתמשים, המטלות שכתבו והמטלות שהם רוצים שייבצעו בDatabase
* הוא מתפעל את המטלות שהמשתמש רוצה שיבוצעו ודואג שישנם שרת עבודה ופועלים שעובדים ביחד על מטלה

פרוטוקול התקשרות בין היחידות בפרוייקט

כל התקשורת נעשית דרך שרתי .bottle המודול bottle של פייתון מאפשר ליצור שרתים סטטיים אשר מקבלים בקשות GET וPOST בroutes השונים שלה לפי פרוטוקול HTTP. העדפתי את bottle מאשר להשתמש במודול sockey כי קל יותר להשתמש בפרוטוקול קיים מאשר להמציא פרוטוקול חדש עם sockets, מה שהיה לוקח לי יותר מחודש מתהליך הפיתוח.

למרות זאת, כפי שציינתי ב"מבוא", בא אתגר עם כך שהשרת הראשי ועבודה הם סטטים ואינם יכולים לשלוח פקודות זה לזה ולפועלים. לכן, ישנו פרוטוקול מיוחד ביניהם שאפרט בפרק זה הפותר את הבעייה.

אחד מהאלמנטים המרכזיים של הפרטוקול הוא הסטטוסים של כל יחידה המתארים את מצבם הנוכחי מספרית:

|  |  |  |  |
| --- | --- | --- | --- |
| שם יחידה/אובייקט | השם של הסטטוס בקוד | המספר של הסטטוס | משמעותו |
| יחידת עבודה | |  | | --- | | untouched | | in\_progress | | |  | | --- | | 0 | | 1 | | |  | | --- | | היחידת עבודה "פנוייה": אף פועל לא עובד על היחידת עבודה הזו וצריך לחשב אותה. | | היחידת עבודה בתהליך חישוב על ידי פועל מסויים | |
| השרת עבודה | |  | | --- | | no\_work | | has\_work | | finished\_work | | |  | | --- | | 0 | | 1 | | 2 | | |  | | --- | | לשרת אין יחידות עבודה "פנויות"(או שעוד לא יצר אותם או שעל כל יחידת עבודה עובד פועל) | | לשרת יש יחידות עבודה והוא עוד עובד על המטלה | | השרת הפסיק לעבוד על המטלה(או שסיים עד הסוף, או שנתקל בבעיה) | |
| הפועל | |  | | --- | | just\_joined | | Waiting | | Working | | shut\_down | | |  | | --- | | 0 | | 1 | | 2 | | 3 | | |  | | --- | | הפועל רק כרגע נרשם לשרת העבודה וקיבל ID | | הפועל מחכה שיהיה יחידת עבודה "פנויה" לעבוד עליה | | הפועל עובד על יחידת עבודה | | התוכנה של הפרוייקט סויימה | |
| תהליך חישוב היחידת העבודה | |  | | --- | | Calculated | | Failed | | Crashed | | |  | | --- | | 1 | | 1- | | 2- | | |  | | --- | | היחידת עבודה חושבה בהצלחה | | הפועל שחישב את יחידת העבודה לא הצליח לחשב אותה(מסיבות טכניות או אישיות למחשב שהוא רץ עליו( | | כאשר הפועל ניסה לחשב את יחידת העבודה בקובץ מטלה, הקובץ העלה שגיאה, לכן חייבים להפסיק את התהליך חישוב. | |
| מטלה | |  | | --- | | untouched | | Started | | in\_progress | | Finished | | Failed | | Crashed | | |  | | --- | | 0 | | 1 | | 2 | | 3 | | -1 | | -2 | | |  | | --- | | המטלה "פנויה": אף שרת עבודה עובד על המטלה הזו כעת | | ישנו כעת שרת עבודה שמויין למטלה | | השרת עבודה התחיל בתהליך חישוב המטלה והתוצאות הראשונות כבר חושבו | | חישוב המטלה סויים | | שרתי העבודה ניסו לחשב את המטלה אך נכשלו באמצע התהליך | | הייתה יחידת עבודה אשר הקריסה את תהליך החישוב | |

ברוב מעברי המידע בין יחידות, כל יחידה מתעדכנת על הסטטוס הנוכחי של האחר ולפיה קובע את פעולתם הבאה. דוגמא לכך היא שכאשר תוכנת הפועל מורצת, לפני שהוא מתחיל לחשב, הוא כל הזמן בודק את הסטטוס של השרת עבודה שהוא צוות איתו לגבי האם יש לו יחידות עבודה(סטטוס "has\_work"), או לא.

משום ששרת העבודה סטטי ואינו יכול לשלוח הודעה לפועל להתחיל לבקש ממנו יחידות עבודה אחרי שיצר את כולם, אז הלקוח שואל כל תקופת זמן קבועה על הסטטוס של השרת עד שיקבל ממנו את הסטטוס "finished\_work", כדי שהוא יידע שהשרת עבודה סיים לעבוד על המטלה ותפקידו נגמר.

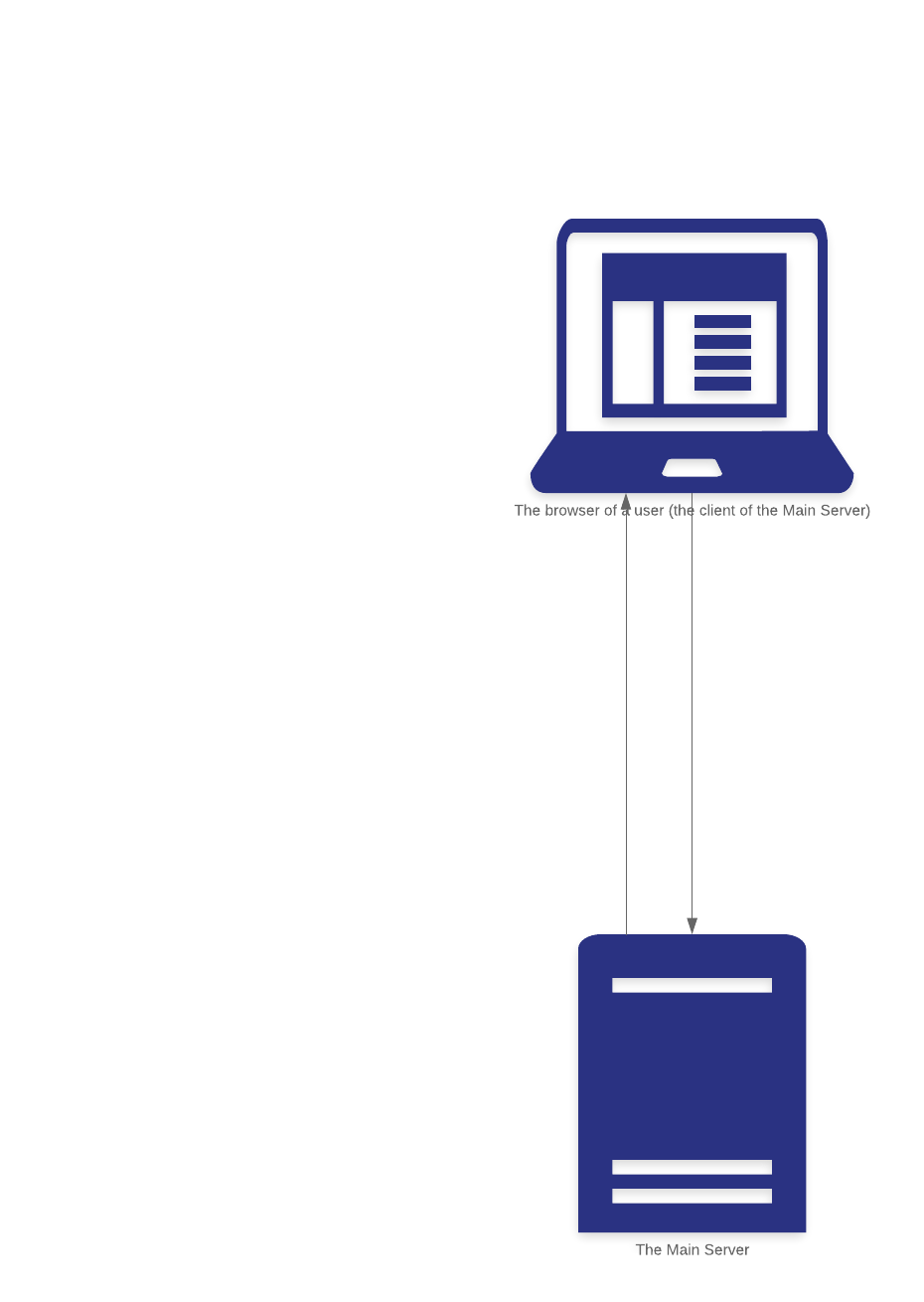
בנוסף לכך, סטטוסים מאפשרים בקרה על התהליכים השונים בכך שמציבים סטטוסים למצבים השונים שלהם. ככה אני אוכל רק להעביר מספר והצד האחר ידע באיזה שלב התהליך של המטלה או חישוב יחידת עבודה תיהיה.

בנוסף לכך ישנו מודול CalcFarmEssiensials, המגדיר פרוטוקול תקשורת נוסף מעל HTTP: אם הלקוח מקבל שגיאת תקשורת מסוימת(excepction במודול התקשורות או Connection Error), הוא יחכה וינסה שוב אחרי זמן קצר(אין טעם מייד לנסות שוב, לכן יחכה קצת שהמצב אצל השרת ישתנה). הוא לפחות ינסה 3 פעמים, ואם בכל זאת הלקוח לא מצליח לתקשר עם השרת, נעלה השגיאה שתקריס את התוכנה(אם זה הפועל או השרת עבודה, הם יפסיקו לרוץ מהשגיאה, אך אם זה השרת הראשי, אין סיבה שבגלל שגיאה אחת אז הוא יפסיק לטפל במטלות של כל המשתמשים ובאתר, לכן הוא ממשיך ומפטר את השרת עבודה שגרם לשגיאה(אפרט על זה אחר כך))

באשר לשגיאות status של HTTP, לרובם הוא יקריס ישירות(כמו 500-האומר שישנו באג בקוד של השרת ולכן אין טעם ללקוח לנסות שוב), אך לכמה בודדים הוא ינסה 3 פעמים(כמו 404- האומר שהשרת לא מוצא משאב שהלקוח דורש ממנו, ולכן יכול להיות שהוא עוד לא הגיע לשרת, אז כדי לנסות שוב אחר כך).

למרות שהתקשורת היא בין לקוחות לשרתים, לפעמים השרתים הם גם "לקוחות" המתקשרים עם שרתים אחרים, כמו שרת העבודה הדרוש מהשרת הראשי מטלות ושולח לשרת הראשי תוצאות.

הפרוטוקול תקשורת בין הלקוח לשרת העבודה



שהלקוח לראשונה פותח את האתר, הוא יוצר חשבון ונותן לאתר את השם משתמש והסיסמה שלו. האתר שולח מהמחשב שלו לשרת הראשי את הפרטי חשבון, והוא יוצר אותם בdatabase.

בתהליך הזה דרוש אבטחה רבה כדי להגן על חשבון הלקוח. אני לא שומר את הסיסמה כמו שהיא, אלא אני מוסיף לו מחזרות אקראית הSALT, מקודד אותה בHASH256 ושומר את הגרסה המוצפנת עם הSALT. ככה, אפילו לי אין את הסיסמה המקורית של המשתמשים וכל פעם שהמשתמש מתחבר, אני מנצל את תכונת החד ערכיות של פונקציות הHASH ומשווה את הסיסמה המקורית עם גרסה של הסיסמה שהוכנסה אשר הוצפנה באותה דרך.

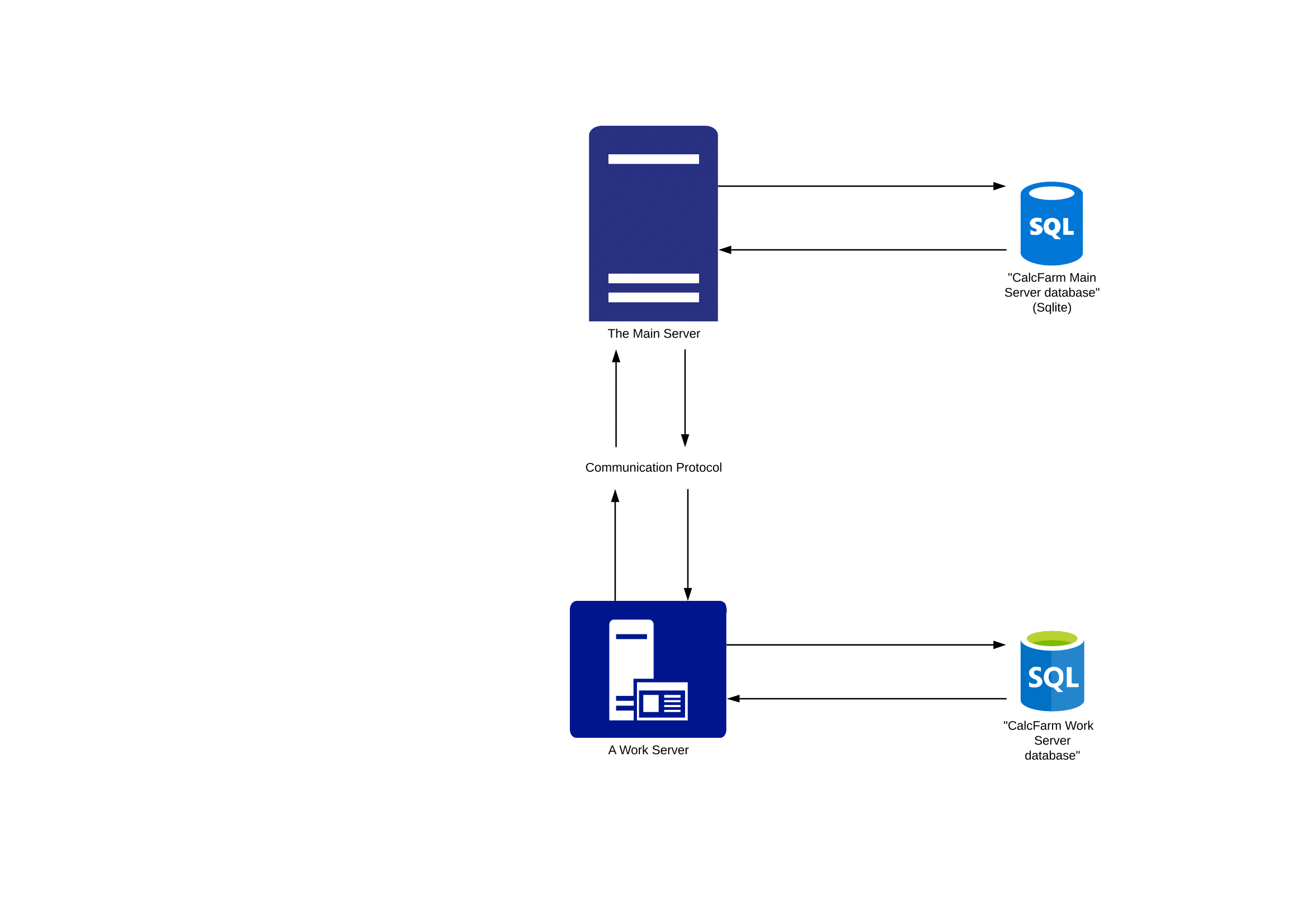
כאשר הלקוח גולש באתר ולוחץ בסרגל הניווט או בלינקים לעמודים שונים, האתר שולח בקשה GET לשרת הראשי, שנותן לו חזרה דפי HTML.

הלקוח גם נתן הוראות מהאתר לשרת הראשי להריץ מטלות מסויימות, או לבטל אותם.

כאשר השרת הראשי מחשב מטלה, הוא שולח לעמודים מסויימים באתר אינדיקציה על התהליך ואחוז ההתקדמות כדי שהלקוח יוכל לבחון ולבקר על התהליך.

בנוסף לכך, השרת הראשי שולח לאתר את התוצאות של המטלה כאשר הוא סיים.

הפרוטוקול תקשורת בין השרת הראשי לשרת עבודה



השרתים בפרוייקט הזה במובן התקשורת הם לרוב סטטים. בעזרת מודול bottle, יש להם routes שמקבילים בקשות http.

חלק מהפרוטוקול ביניהם הוא שכאשר שרת העבדוה ניגש לroute בשרת הראשי, הוא נותן לו כארגומנט ראשון את שם המשתמש שלו כך שהשרת הראשי יוכל לזהות אותו לפי שם המשתמש וה-IP של שרת העבודה.

דבר ראשון ששרת העבודה עושה שהוא הופעל, הוא ניגש לשרת הראשי, נותן לו את שם המשתמש שהריץ אותו והכתובת IP שלו ומבקש אם יש לשרת הראשי מטלות שהמשתמש שלו רוצה שייבצעו כעט ה"פנויות"(מטלות בעל סטטוס "Untouched").

אם יש לו מטלה, אז השרת הראשי נותן לו את פרטי המטלה ומממן אותו להיות אחראי על חישובה וכל פועל שהשרת הראשי יחליט שיעבוד על המטלה הזו יעבדו עם השרת עבודה הזה.

כאשר שרת העבודה מתחיל לעבוד על המטלה, הוא שולח לשרת עבודה עדכון כל פעם שקיבל תוצאות של יחידת עבודה עם אחוז עבודה שכבר עשה(יותר פירוט ב"אלגוריתמים") ופרטים על יחידת העבודה והתוצאות שלה.

השרת הראשי רושם את כל יחידות העבודה בטבלה שיצר למטלה הזו ורושם את בשתי קבצים: בקובץ "Work\_Units" שבו ישנו פרטים על כל יחידת עבודה ובקובץ "Results" בו השרת אוטומטית מחבר את התוצאות מכל יחידות העבודה לרשימה ענקית(אם התוצאות הם גם רשימות, אז הוא אוטומטית יחבר את הרשימות זו לזו). כאשר המטלה היא תנאית, בקובץ "results" יירשם רק תוצאות היחידת עבודה שבה התנאי התקיימה או הופרה וכאשר יחידת עבודה הקריסה עובד, היחידת עבודה תירשם בנפרד מהשאר.

השרת הראשי שומר על יחידות העבודה בסדר בכך שמצרף אותם לפי סדר הID שלהם, עד "יחידת העבודה העוקבת האחרונה".

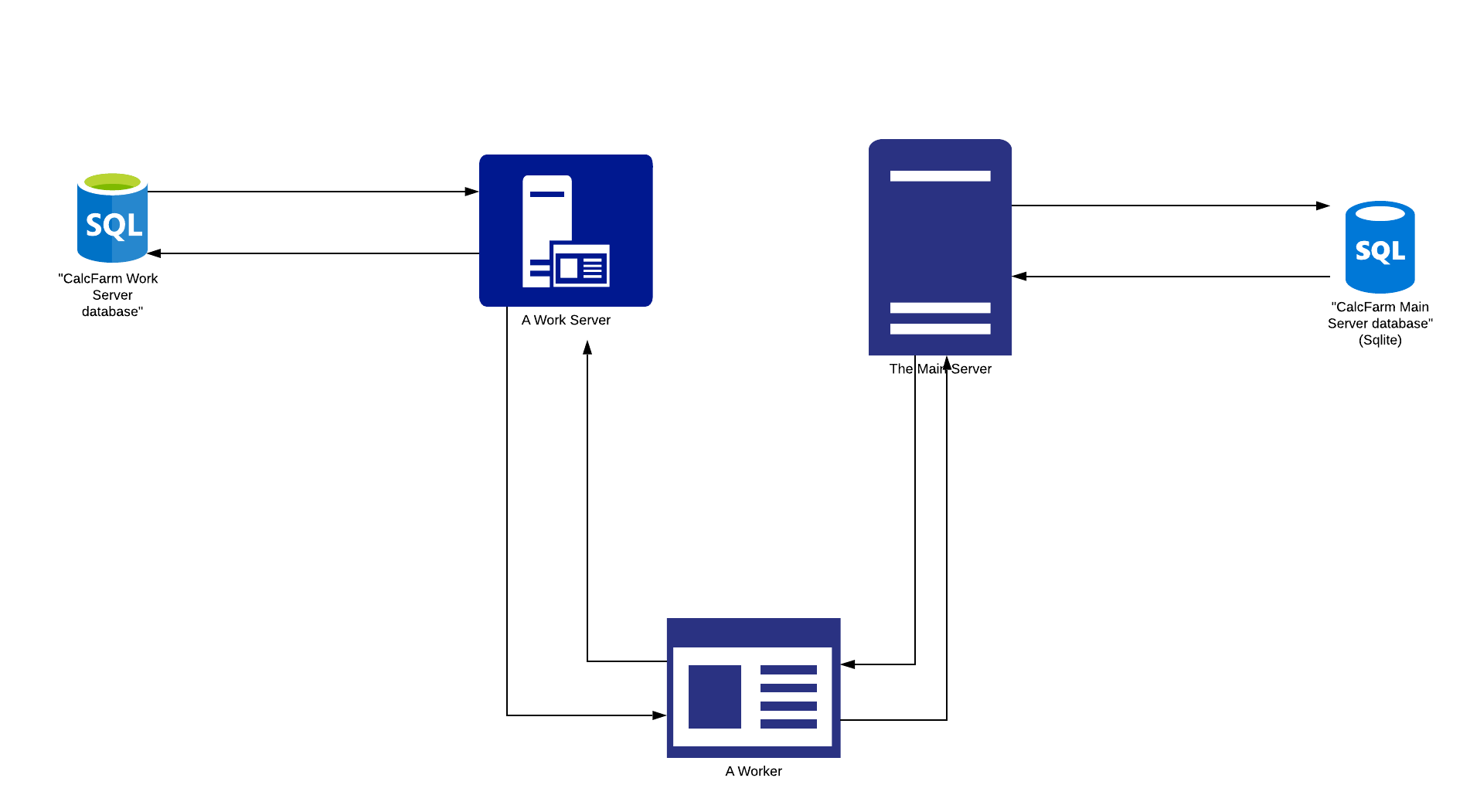
"יחידת העבודה העוקבת האחרונה" היא יחידת העבודה עם ה-ID הכי גדול ככה שיש לשרת הראשי את כל יחידות העבודה, מהראשונה עם ID של 1, עד ליחידת העבודה הזו, ברצף עוקב. לדוגמא, אם השרת קיבל את כל ה10 יחידות עבודה הראשונות, אז קיבל את יחידת עבודה מספר 12, הוא לא יצרף אותה או את יחידת עבודה מספר 13 או 14 לקבצים, כי 12 אינו "עוקב" ל10 והאוסף יחידות עבודה אינו רציף. חסר יחידת עבודה מספר 11.

ככה יש לו אוסף רציף של כל התוצאות.

ה-"יחידת העבודה העוקבת האחרונה" גם משמשת כנקודת מוצא שאם משהו השתבש בתהליך חישוב המטלה: בין אם השרת עבודה לא מגיב לזמן מה או קרס, אז השרת הראשי יודע שכל המספרים בין 1 למספר האחרון ב-"יחידת העבודה העוקבת האחרונה" טופלו (כי יש לו את כל יחידות העבודה בסדר עוקב).

כאשר שרת העבודה סיים את המטלה, הוא שולח לשרת דוח הכולל את הזמן שלקח לשרת עבודה לבצע את המטלה ואת הסטטוס של התהליך: בין אם הצליח(סטטוס "finished") או שקרס(סטטוס "crashed") או שלא הצליח (סטטוס "failed").

הפרוטוקול תקשורת בין הלקוח לשרת הראשי ולשרת עבודה



הפרוטוקול תקשורת בין פועל לשרת הראשי

ראשית כאשר התוכנה של הפועל התחילה לרוץ, הוא מתחבר לשרת הראשי ומבקש לעבוד על מטלה של המשתמש שלו. אם ישנם מטלות של המשתמש ששרת עבודה כבר עובד עליו ולא סיים, הוא יבחר מטלה מסויימת(לפי האילוצים של אחוזי כוח העבודה הדרוש וכמות הפועלים בכל מטלה) וייתן לו את הIP של השרת עבודה כדי שיוכל להצטרף אליו.

הפרוטוקול תקשורת בין פועל לשרת עבודה

הוא משתמש בIP של השרת עבודה כדי להתחבר אליו. הוא מקבל ממנוID יחודיי הרשום בDB של השרת עבודה.

הפועל כל זמן מתחבר לשרת עם הID ומבקש מהשרת יחידות עבודה לחשב.

כפי שהדגמתי בתחילת הפרק, הפועל כל הזמן מבקש מהשרת עבודה יחידות עבודה לחשב:

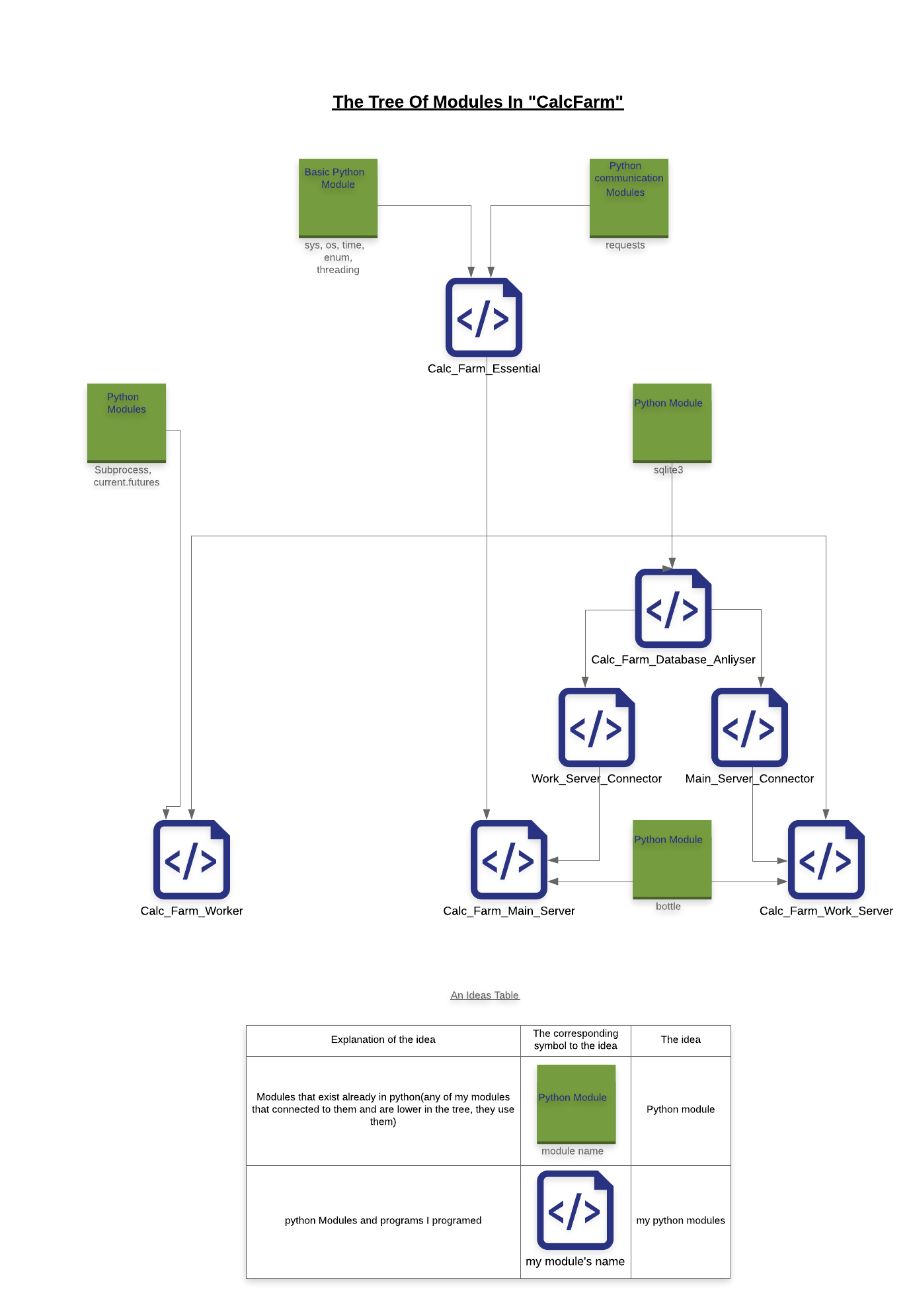
אם הלקוח קיבל מהשרת שאין לו יחידות עבודה, הוא יחכה זמן קצר וייבקש שוב ושוב עד שייקבל יחידת עבודה.

אם יש לשרת, הוא שולח לו את התחום המספרים של יחידת העבודה שיחשב. בנוסף לכך העובד ידרוש מהשרת משאבים שהוא צריך בשביל לחשב כאשר חסר לו כמו הקובץ פייתון של המטלה שאיתו הוא מחשב את היחידת עבודה.

הסכם קבוע ביניהם בפרוטוקול הוא שהפועל תמיד שולח לכל route כארגומנט ראשון את הID שלו כדי שהשרת עבודה יוכל לזהות אותו.

כאשר העובד סיים לחשב יחידת עבודה, הוא שולח את התוצאות לשרת, הרושם את התוצאות של יחידת העבודה ואת התחום שלה בדוח שהוא שולח לשרת הראשי ומסיר את יחידת העבודה מהטבלה של יחידות העבודה שממנה הוא נותן לפועלים(אין סיבה שהשרת עבודה יעבור על כל ה9999 יחידות עבודה שכבר סיים רק כדי למצוא את היחידת עבודה היחידה שנותרה).

בנוסף לתוצאות של יחידת עבודה, הפועל שולח לו את הסטטוס של החישוב אצל הפועל, בין אם הוא הצליח, נכשל(בין אם שגיאה במחשב או בהרצת הקובץ) או שהיחידה הקריסה את הקובץ.

המודולים בפרויקט 

המודולים שאני כתבתי:

* CalcFarm\_Database\_Analyser- זהו מודול מיוחד המבוסס על מודול של Sqlite3 מפייתון שמקל על תכנות בsqlite. זהו פרויקט מעצמו:

אני פישטתי את sql לפונקציות שכמעט לא דורשות קוד sqlite כמו:

"create\_table", אשר יוצר טבלה חדשה אחרי שבדק אם קיים שולחן אם אותו שום בDatabase,

"info\_table", אשר מקבלת שם טבלה ומחזירה מידע על הטבלה: tuple שבאחד מילון "מתורגם" של כל העמודות, שבו לכל שם של עמודה משויך tuple עם מידע על העמודה הזו.

והפונקציה שהשתמשתי בה הכי הרבה: "find\_specific\_record", אשר מחפשת שורות אשר בעל ערכים ספציפסים בעמודות ספצפיות. היא מקבלת את שם טבלה, ומילון שבו כל שם עמודה היא מפתח לערך שאנחנו רוצים לבדוק בה.

לדוגמא:

find\_specific\_record(“Students”, {“first\_name”: “Georgie”, “last\_name: ”Lester”})

יש גם את הארגומנט return\_data, אשר אם הוא שווה לTrue, אז הפונקציה גם תחזיר את השורות בעל ערכים הללו ו"תתרגם" אותם, אך אם אתם רוצים רק לבדוק שקיימים שורות בעל ערכים כאלה ולא רוצים לבזבז זמן "לתרגם", אז שreutrn\_data יהיה שווה לfalse והוא יחזיר True או False אם קיימים שורות.

במודול אתה יוצר אובייקט “Database”, אשר אחראי לתקשר עם מבנה הנתונים וממנו אתם קוראים לפונקציות לבצע על מבנה הנתונים. בעוד שבמודול של Sqlite3, צריך כל פעם לפתח ולסגור תקשורת עם מבנה הנתונים, המודול שלי עושה את זה אוטומטית.

בנוסף לכך, למודול יש features אשר מייעלים ועוזרים יותר, כמו "תרגום" שורות: המרת quarry של שורה המתקבלת במודול הרגיל כ-tuple של ערכים למילון שבו כל עמודה היא מפתח לערך הנשמר בעמודה בשורה הזו, ככה אתה לא צריך לזכור את הסדר של עמודות או ליצור ENUM לכל טבלה; ואם מתקשר רק שורה אחת מה-quarry, אז התוכנה לא מחזירה אותו כרשימה של איבר אחד, אלא פשוט מחזירה את הtuple של השורה היחידה, או אם לא נחזר quarry ריק ולא נמצאו שורות, אז הוא לא יחזיר רשימה ריקה, אלא פשוט יחזיר None. הוא גם תומך בthreading: אם אתה נותן לאובייקט database מנעול, אז הוא ידאג ששני threads לא יתקשרו עם הdatabase בו זמנית.

הfeature- הכי חשוב שהוספתי שמייחד אותו וקריטי לפרויקט שלי הוא שיש למודול את היכולת לשמור סוגי נתונים שפייתון תומך בהם אך SQL לא כמו רשימות ומילונים. איך שהמודול שומר אותם הוא על ידי המרתם לבתים וחזרה דרך מודול pickle. כחלק מהפרוטוקול של המודול, העמודות בהם נשמר נתונים מחייבים להיות מסוג הנתונים “Longblob” כדי שהמודול ידע לקודד ולתרגם חזרה את העמודות הללו.

* CalcFarm\_Essentials- זהו מודול הכולל את כל הפונקציות ומשתנים הנחוצים לשלושתם, כולל פונקציות לתקשורת שכל 3 היחידות של הפרויקט משתמשות כדי לתקשר זה עם זה ואת פרוטוקול הסטטוסים(יותר מידע על איך הוא עובד בתחילת הפרק "פרוטוקול תקשורת").
* CalcFarm\_Main\_Server\_db\_Conncetor- המודול הזה הוא הרחבה של תוכנת השרת הראשי.

יש בה פונקציות וקבועים של השרת הראשי, כך שבתוכנת השרת הראשי יש פחות קוד והכי חשוב, היא אחראית על התקשורת בין מבנה הנתונים לשרת: יש בה את כל הפונקציות שבהם השרת הראשי כותב וקורא מידע מהטבלאות.

CalcFarm\_Worker\_Server\_db\_Conncetor- המודול הזה זהה בתפקידו לזה של CalcFarm\_Main\_Server\_db\_Conncetor רק שהוא מרחיב את תוכנת שרת העבודה.

אלגוריתמים:

* איך השרת מחלק מטלה ליחידות עבודה ואיך הוא מנהל אותם:

השרת מקבל את תחום המספרים הכללי מהשרת הראשי. ראשית, הוא מחלק את תחום המספרים הכללי לתחומים קטנים יותר בעל כמות מספרים קבועה הנקראים "יחידות עבודה".

לדוגמא אם התחום מספרים הכללי הוא בין 1 ל100 ובכול יחידת עבודה יש 10 מספרים, אז התחום יחולק לתחומים 1 עד 10, 11 עד 20, 21 עד 30... ו91 עד 100(כמובן שאין טעם שיהיה כמות כזו קטנה ביחידת עבודה כי קבצים הרצים על מחשבים שיש היום מסיימים במילישנייה והאמת היא שלוקח יותר זמן כאשר יש יותר יחידות עבודה. לכן עדיף שבכל יחידת עבודה יהיה כמות גדולה אך לא גדולה מדי כי המטרה של התוכנית היא לחלק תחומים גדולים)

ככה, פועלים הרצים על מחשבים מהירים יותר וכך יכולים לחשב יותר יחידות עבודה, ידרשו יותר יחידות עבודה כאשר סיימו והיחידות עבודה "יזרמו" אליהם כמו מים וישנה דינאמיות בחלוקת יחידות העבודה.

כל היחידות עבודה נשמרות בטבלה "Work\_Units" עם סטטוס המתאר את מצב חישוב יחידת העבודה:

0=Untouched- אף פועל לא עובד על יחידת העבודה הזו

1=in\_progress- ישנו פועל מסוים אשר לקח את יחידת העבודה הזו ומחשב אותה

כאשר פועל מבקש יחידת עבודה, השרת מחפש עם פקודת select יחידות עבודה בעל סטטוס 0 ומחזיר את הראשונה שמוצא ושהשרת מקבל ממנו את התוצאות של יחידת עבודה, הוא מוחק אותו מ"Work\_Units" ומעביר אותו עם התוצאות שלו לטבלה "Work\_Unit\_Results"(זה לא הגיוני שהפקודת select תעבור על כל 9999 יחידות העבודה שכבר סיים רק כדי להחזיר לפועל את ה1 שלא נגעו בה).

* איך השרת עבודה מחלק יחידות עבודה לפועלים.

לכל יחידת עבודה יש כמה מצבים:

untouched = 0-משמע שאף פועל לא עובד על יחידת העבודה הזו  
in\_progress = 1 – משמע שפועל לקח אותו בתהליך חישובה  
finished = 2 - משמע שיחידת העבודה כבר חושבה ותוצאותיה שמורות בdb

כאשר יש לשרת יחידות עבודה, זה כמו בופה בחינם של אוכל כאשר יש לו תור. המנהל בופה מסיר את הסרט האדום, פותח את העסק ונותן לאורחים לקחת חתיכת אוכל אחד אחר אחד: הראשון ממלא את הצלחת שלו, ואז השני אחריו ממלא את הצלחת שלו וכו... עד שנגמר האוכל, ואז המנהל בופה מונע גישה לשולחן, ושם סרט אדום וסוגר את הבופה(לא הייתי רוצה להיות זה שחיכה שעות בתור רק כדי שישאר רעב).

במקרה הזה המנהל בופה הוא השרת עבודה, הבופה הוא הטבלה בה שמור יחידות העבודה בDB והאורחים הרעבים הם הפועלים.

כאשר פועל "רעב" ליחידת עבודה

אני קורה לאלגוריתם "free for all"/בופה חופשי

האמת היא שהאלגוריתם הזו דומה לאיך שהשרת המרכזי נותן מטלות לשרתי עבודה.

* השרת הראשי מחלק את העובדים בין המטלות של המשתמש המשויך אליהן בדרך הבאה:

קודם כל, הוא מוצא את המטלה בעל הכמות המינימלית של פועלים העובדים עליה ומחזיר לו את המטלה עם הip של השרת עבודה המטפל במטלה. אם לכמה מטלות יש את אותו ערך מינימלי(דוגמה טריוויאלית תהיה שלכולם יש 0 עובדים), השרת יבחר באקראיות אחד מהם וישייך אותו אליו כדי שכל השרתים יקבלו עובדים והם יחולקו באופן שווה.

ישנו את האפשרות לציין במטלה את אחוז העובדים מתוך הכמות עובדים הכללית המשוייכים למשתמש שיעבדו אליה וככה להגדיר את סדר העדיפויות של המטלות ואיזה מהן צריכה יותר פועלים. לדוגמא אם יש 5 מטלות, 10 עובדים התחברו ולאחד מהם נקבע שהוא צריך 20% כוח עובדים, אז לו יהיה 2 עובדים ולאחד שנקבע 50% יהיה 5 עובדים אפילו אם זה אומר שלחלק מהמטלות לא יהיו עובדים, כי הם לא בעל אותה עדיפות כמוהם(למטלות בעל אחוז יש עדיפות גבוהה יותר מלמטלות בלי אחוז)

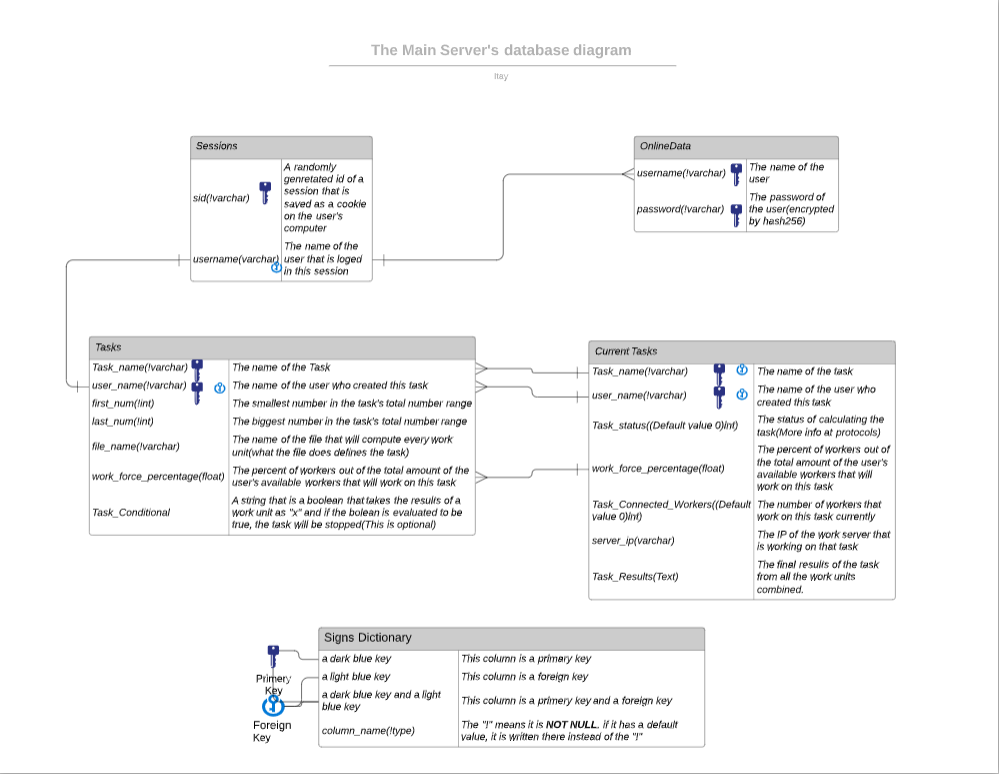
כאשר עובד חדש של משתמש מבקש מהשרת הראשי לעבוד על מטלה, הוא ייתן לו מטלה ככה שישמרו האחוזים: הוא ייחשב את האחוז כפול כמות הפועלים הכללית הנוכחית של המשתמש ובודק בכל מטלה האם מספר הפועלים שלה קטן ב-1 מהמספר, אם כן אז היא תוסיף לו. כאשר האחוז לא מדוייק, הוא ידלג על המטלה הזו וייתן למטלה אחרת. ככה הכמות נשמרת מדוייקת לאחוז ובפעמים שבהם המספר לא יוצא שלם, הפועלים מפוזרים גם למטלות אחרות.

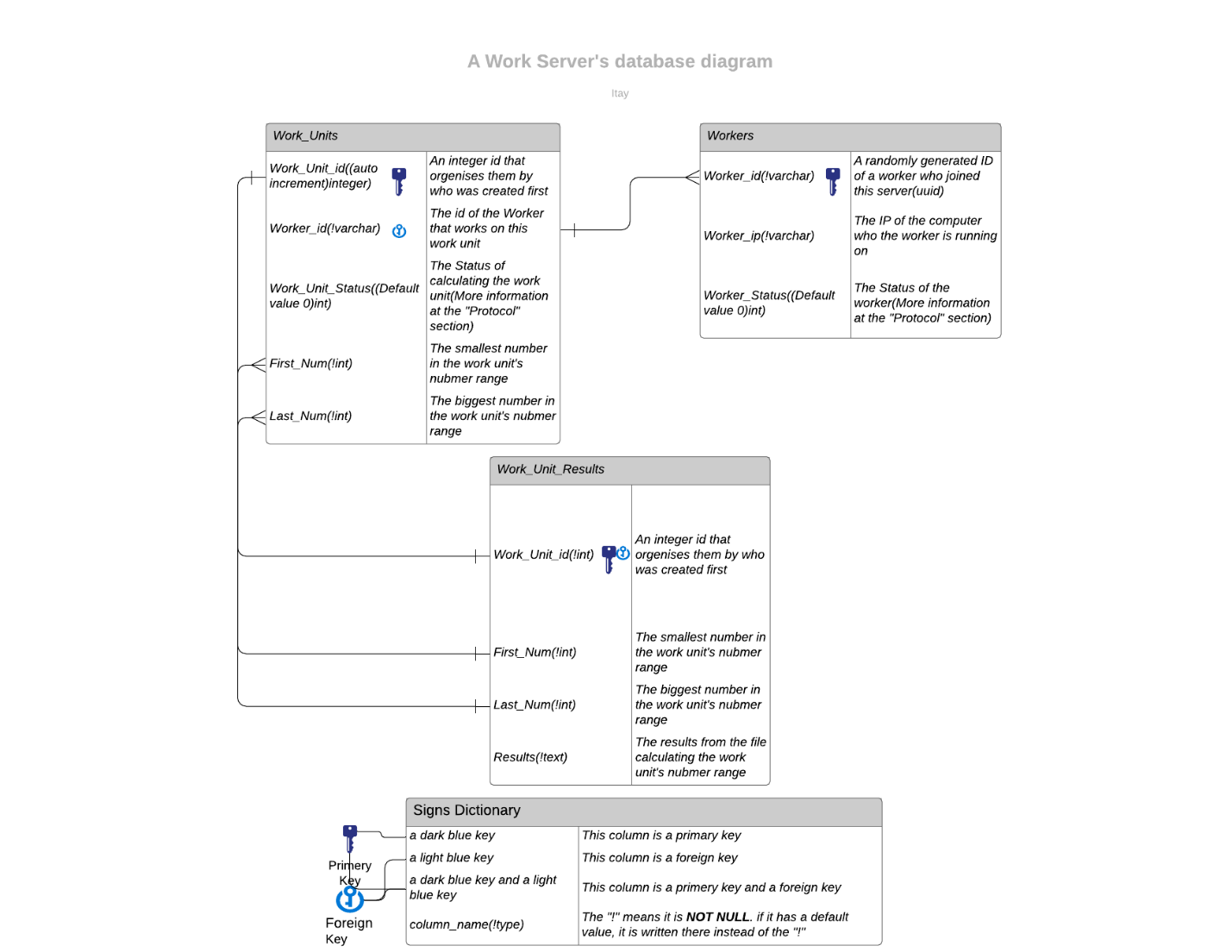
בנוסף לכך, אם יש גם מטלות שאין להם אחוז, הפועל יקבל את השרת בעל הכי פחות פועלים העובדים עליה. ככה העובדים יחולקו ביניהם באופן שווה. לדוגמא נגיד שלכול המטלות בלי אחוז יש 0 פועלים, פועל הצטרף ומוין למטלה הראשונה שבלי אחוז. אז לו יש פועל אחד ולשאר יש 0 פועלים, אז הפועל הבא ימויין לאחד מאלה ולא מאותו אחד כי הוא כבר לא בעל הכמות הכי קטנה של פועלים. וכן הלאה, ככה שיהיה כמות שווה של פועלים בכל המטלות בלי אחוז. אם השרת הראשי עובד על מטלות עם ובלי אחוז, אז האחוז הנותר מסכום האחוזים שלהם של פועלים יפוזר בין המטלות בלי אחוז. לדוגמא, אם יש מטלה של 30% ו3 מטלות בלי אחוז, אז הפועלים קודם יפוזרו בין המטלות בלי האחוז כי הם צורכות בכללי 70% מכמות הפועלים.

* כאשר העובד סיים לחשב יחידת עבודה, הוא שולח את התוצאות לשרת, הרושם את התוצאות של יחידת העבודה ואת התחום שלה בטבלה Work\_Unit\_Resultsומסיר את יחידת העבודה מהטבלה של יחידות העבודה שממנה הוא נותן לפועלים(אין סיבה שהשרת עבודה יעבור על כל ה9999 יחידות עבודה שכבר סיים רק כדי למצוא את היחידת עבודה
* השרת עבודה גם מחשב את אחוז העבודה שהספיק עד כה על פי כמות יחידות עבודה שחושבו עד כה כבר הפועלים שלו חישבו מתוך הכמות הכללית של יחידות עבודה:

**מבנה נתונים**

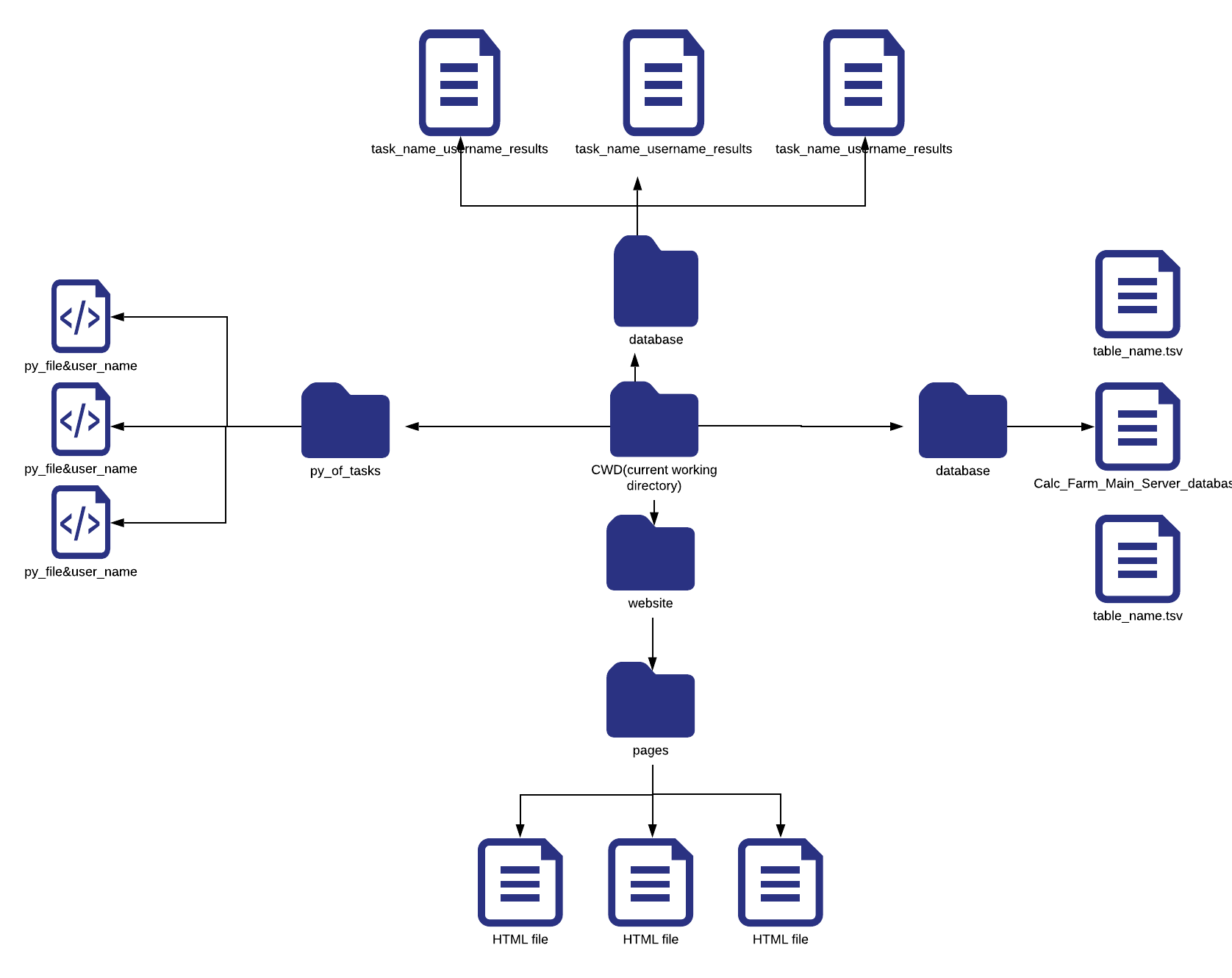
אני משתמש במבנה הנתונים SQLITE ובמודול בפייתון SQLITE3 כדי לתקשר איתו. חלק גדול מהפרויקט הוא מודול מיוחד שאני תיכנתי שמפשט את האינטראקציות עם מבנה הנתונים(יותר מידע בפרק הארכיטקטורה בתת פרק על מודולים)





**מדריך למפתח**

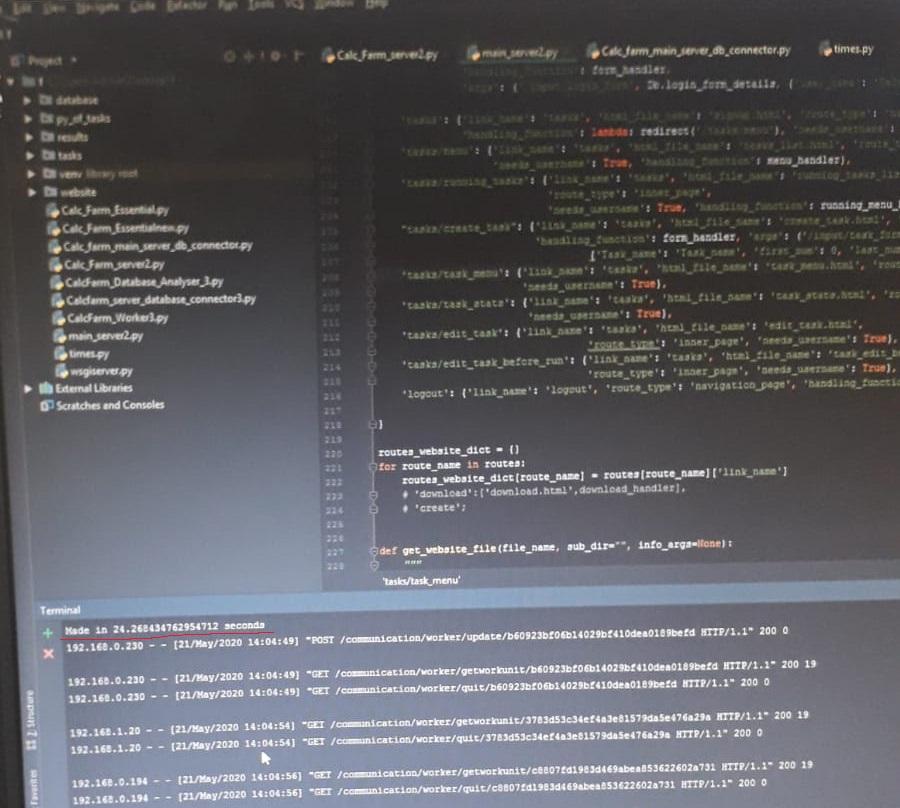
השרת הראשי

**סוגי הבדיקות שעשיתם**

במהלך פיתוח הפרויקט, הרצתי את הפרויקט מלא פעמים: אחרי כל feature שהוספתי, הרצתי את הפרויקט כמה פעמים, ועשיתי debugging.

הנה כמה דוגמאות לבדיקות מיוחדות:

1. כאשר הייתה האפשרות לגשת למעבדות אחרי תקופת הבידוד, ניסיתי להריץ את הפרויקט עם כמה שיותר מחשבים מחוברים כאשר כל מחשב הריץ פועל אחד: ניסיתי עם פועל אחד(לקח 67 שניות), עם שתי פועלים(לקח 33 שניות), עם שלושה פועלים(לקח 24 שניות),



עם ארבעה פועלים(לקח 21 שניות) ועם חמישה פועלים (לקח 19 שניות). בנפרד לכך, אני בדקתי מה קורה כאשר מכל מחשב מריצים 4 פועלים דרך multiprocessing . הרצתי על שלושה מחשבים(12 פועלים בכללית על מטלה אחת) ויצא 8 שניות.

1. בדקתי את האינטרקציות בין היחידות השונות בנפרד: לפעמים אני הרצתי את הפועל והשרת עבודה עוד לפני שהמשתמש הריץ דרך האתר מטלה כדי לראות האם הם יתחילו לעבוד, לפעמים כבר בחרתי ללריץ מטלה מסוימת מהקוד ולא מהאתר כי האתר תקע את הפרויקט.
2. בדיקה רבה שרצתי על המחשבים שהפעלתי עליהם גם את הפרויקט את התוכנות “bottle\_server” , השרת bottle הכי פשוט שאתם יכולים לדמיין ו”bottle\_client”, שפשוט ניגש לאחד מה-routes של השרת. אני לא רק בדקתי שהמחשבים מצליחים לתקשר זה עם זה באמצעות bottle ואם הבעיה בתקשורת ולא בפרויקט, אלא גם בדקתי עליהם את הפונקציות תקשורת המורכבות(המצויינות בפרק על פרוטוקול תקשורת).

**סיכום/רפלקציה**

היה לי מאוד כיף לעבוד על הפרוייקט הזה. המחשבה שאני יוצר כלי חישוב עוצמתי עם פונטנציאל אדיר דחף אותי קדימה.

למרות זאת, זה לא תמיד היה פשוט.

בהתחלה האתגר הכי גדול שלי היה תכנון הפרוייקט: נרתעתי מהאבסטרקטיות ולא היה תמונה מפוקסת של מה הפרויקט יהיה, מהו מטלה, איך הוא יעבוד, איך זה יראה. הרגשתי כאילו אני משוטט בחושך ושאני לא יודע איך לממש את הרעיון.

היו כמה דברים שעזרו לי:

כאשר הייתי צריך לחשוב על רעיון מורכב, נתקלתי בבעיה רצינית , תכנתתי אספקט מסוים או לחשוב איך הוא יעבוד, אני ידעתי שאם אשאר תקוע במעבדת מחשבים עם הלחץ והמחשבות הללו, אני לא אתקדם. לכן אני הייתי יוצא מהמעבדה ועושה הליכה קצרה. ההליכה קודם כל הייתה מרעננת ונתנה לי אנדרנלין שעורר את הגוף והמוח והעובדה שנעתי ולא נשארתי במקום עזרה לי להמשיך קדימה.

בנוסף לכך, ככל שעבדתי על הפרוייקט יותר ויותר, חשבתי עליו יותר, דיברתי עליו עם אחרים יותר, הרגשתי איתו יותר ויותר נוח. למדתי שברגע שאני לא בטוח לגבי משהו, תשחק איתו, get a feel, כפי שאומרים.

שתיכנתתי את הפרויקט אני רק תיארתי לעצמי כמות פועל ושרת שמחלק עבודה, אך כעת, אני תומך בכמה שרתים שכל אחד מחלק מטלה בו זמנית, משתמשים יכולים ליצור מטלה ולראות את כל התהליך באתר מיוחד. בהחלט

אני לוקח איתי הרבה דברים מתהליך הפרויקט הזה: איך לעבוד על פרוייקט ענקי, איך לעבוד עם משקיעים ואנשים שסומכים עליך, איך להתמודד במצב שאין לך את כל התשובות ואתה תקוע, ואיך לתכנן תקשורת ורשתות.

עם זאת, אני גם למדתי הרבה לקחים:

אני מאמין שהייתי צריך לעשות יותר מחקר.

אני מאמין שהייתי צריך לתאר יותר את הרעיון לפרטים: איך אני רוצה שהשרת יהיה? מה כל הfeatures שאני רוצה. בנוסף הייתי צריך כל פעם שהייתי מוסיף feature לבדוק איך היא משפיעה על השאר המרכיבים

**באגים פתוחים / בעיות קיימות + מגבלות שקיימות**ישנם בעיות כמו עם התקשורת, שהאתר איכשהוא מקפיא את השרת ומונע ממנו לתקשר עם היחידות האחרות.

לא הספקתי לטפל בstatus errors, כמו כאשר שהמשתמש מכניס פרטים לא נכונים לlog in, אין דף מיוחד המראה לו שהכניס פרטים לא נכונים

**הרחבות שהייתי רוצה להשלים בהמשך הפרויקט**

ישנם גם features שלא הספקתי:

1. האפשרות להעלות את הקובץ פייתון שלך למטלה חדשה
2. הייתי מוסיף את האפשרות להוריד את התוכנות של הפועל ושרת העבודה במקום שאצטרך להתקין אותם על כל מחשב באופן מהיר
3. שיהיה domain לשרת הראשי, כך אוכל להריץ אותו על כל מחשב מבלי לזכור את הIP שלו ולהעתיק אותו לשאר התוכנות

ביבליוגרפיה:

מקורות אינטרנט:

* מדריך לטכנולוגיה של Charity Engine <https://docs.google.com/document/d/18UroLy_2zGr8z6tur2HO7EOGJA-WrGDmtNY6WnI-_Cc/edit#heading=h.252aed1lrucc>

כתבה על 42: <https://www.eurekalert.org/pub_releases/2019-09/uob-sot090619.php>