**EASS שיעור 6- תכנון הפרוייקט, חידוד על Dockerfile ובדיקות**

* Draw.io - אתר שאיתו ניתן ליצור דיאגרמות בלוקים
* Load balancer – מציג ללקוח נקודת חיבור (כתובת VIP- Virtual IP) אחת למערכת ואז כשהלקוח ניגש לנקודת החיבור הזו ה-load balancer קובע לאיזו אפלקציה מסוימת לשלוח את החיבור (במקום לשלוח לשרת עצמו). באופן כללי, הוא מהווה reverse proxy. מוסיף גם אלמנט של אבטחה.
* DNS- פרוטוקול שמבצע המרה בין שם מילולי של רכיב לכתובת IP שלו.
* Caching – פעולת אחסון קבצים ומידע (שהשתמשו בהם לאחרונה) ברכיב אחסון זמני שניתן לגשת אליו במהירות.
* Replica – תהליך שכולל שיתוף מידע שמבטיח אחידות במערכת. דוגמאות:

1. Data replication, where the same data is stored on multiple [storage devices](https://en.wikipedia.org/wiki/Data_storage_device)
2. Computation replication, where the same computing task is executed many times. Computational tasks may be:

* Replicated in space, where tasks are executed on separate devices
* Replicated in time, where tasks are executed repeatedly on a single device
* Blue-green deployment – אפשרות לשנות ולהוסיף עדכונים עם פחות downtime (כלומר, ללא זמן שבו הלקוחות לא יכולים לגשת למערכת) ועם פחות סיכונים.

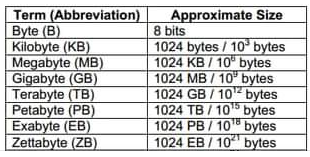
כיצד נתכנן מערכת?

* להבין מי הם הלקוחות
* איך הלקוחות יוכלו להשתמש במערכת?
* מה מנסים למכור? מה המטרה של המערכת?
* מה הקלט והפלט של המערכת?
* נבנה דיאגרמת בלוקים למערכת - חשוב שנתקבע בהתחלה על הארכיטקטורה של המערכת.
* נכנס לעובי הקורה עבור כל service בהתאם לסוג שלו. לדוגמה:

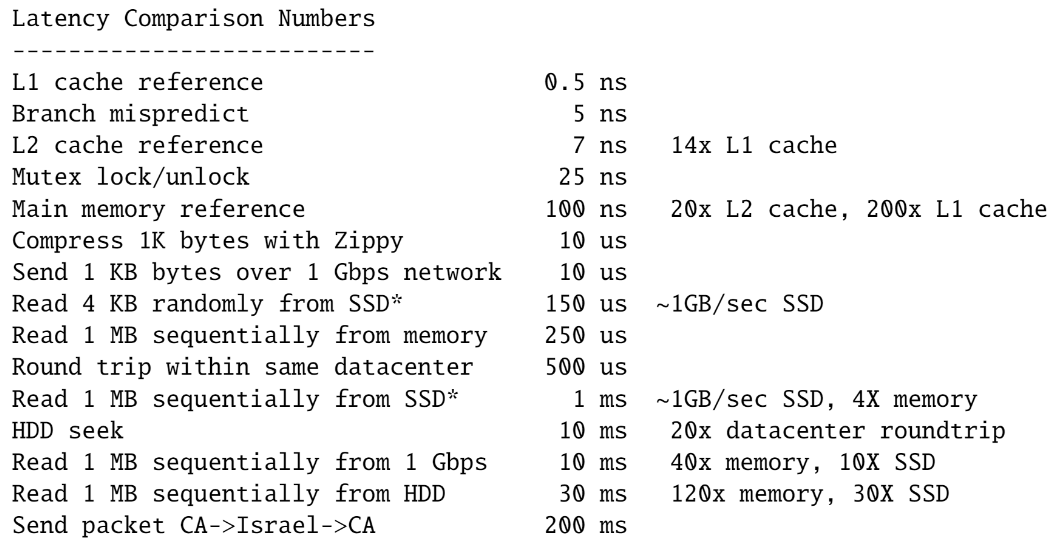
האם צריך בסיס נתונים? איזה סוג? האם צריך יותר מבסיס נתונים אחד?

* מתחילים לבנות רכיב אחר רכיב, בצעדים קטנים, את המערכת. תחילה נבנה את המערכת עבור לקוח אחד בפשטות. בהצשך נבצע הרחבה לפי הצורך. משמעות הדבר היא לא להתקבע על מקרי קצה מסויימים אלא לתכנן את המערכת הכללית- השלד הבסיסי הנדרש בהתאם למטרת המערכת.
* Scaling – התאמת המערכת לכמות לקוחות גדולה יותר, התאמה לעומסים, הוספת replica , הוספת load balancer , שימוש ב-caching וכו'
* ביצוע בדיקות (טסטים)

יחידות מידה נפוצות:

סדרי גודל של זמן לביצוע פעולות:



**חידודים בנושא Dockerfile:**

* פקודות כמו RUN, COPY, WORKDIR הן פקודות שיוצאות לפועל בשלב ה-docker build של האימג'. פעולות שרשומות תחת פקודת CMD הן פעולות שיוצאות לפועל רק אחרי שמבצעים docker run לאימג' (שנוצר ע"י ה-Dockerfile הזה).
* WORKDIR – פקודה שמטרתה לשנות את המיקום שבו הפקודות הבאות ירוצו. במידה והמיקום שניתן לה לא קיים היא יוצרת אותו. לדוגמה:

WORKDIR /app

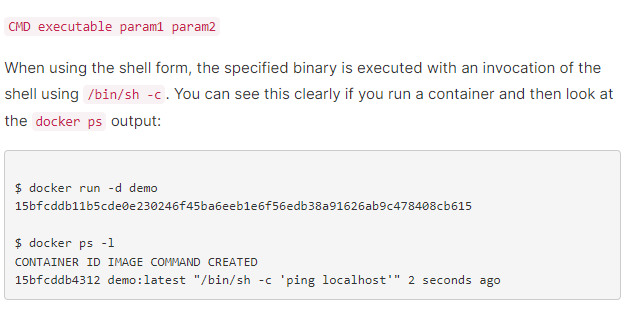
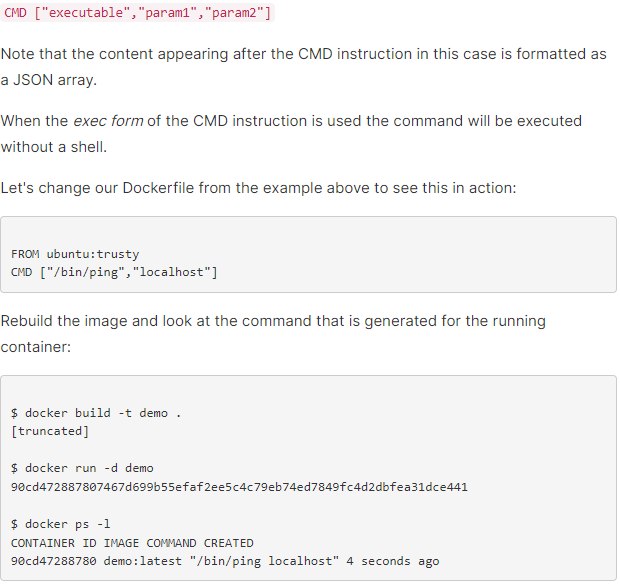
COPY . .

CMD ["uvicorn","main:app", …………….]

הפקודה WORKDIR מעבירה את המיקום לתיקיית app.

בפקודה הבאה COPY . . המיקום שאליו מעתיקים את הקבצים הוא /app (כי כעת הנקודה השנייה הפכה להיות /app)

* ל-CMD יש שתי צורות כתיבה:

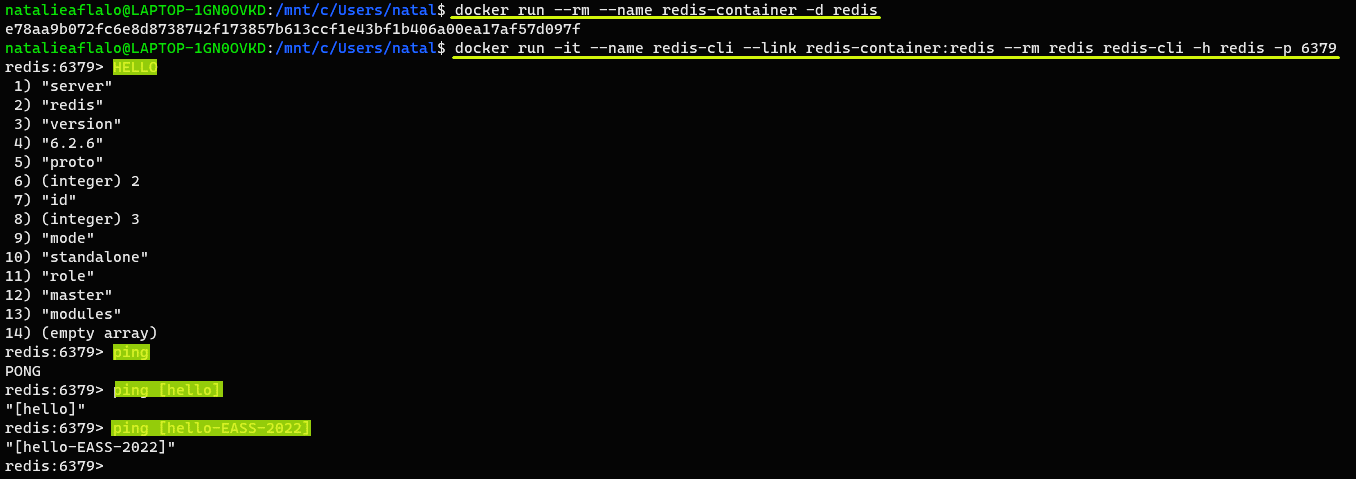
1. Shell – כתיבה פשוטה של המשפט אותו נרצה להריץ. המשפט ירוץ אוטומטית באמצעות /bin/sh  
   
2. Exec – כתיבת הפקודה כשכל מילה היא string במערך שסביבה מרכאות. המשפט ירוץ באמצעות מה שאנחנו בוחרים  
   

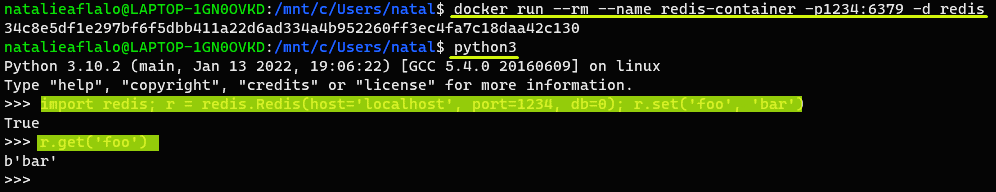
* docker logs dockerID – מציג מידע מורחב על container שרץ כרגע בהינתן ID או שם של הcontainer. לדוגמה, מציג הדפסות שרשומות ב-Dockerfile (הן לא מוחזרות למשתמש)

**Database מידע כללי:**

* מסד נתונים גרפי הוא בסיס נתונים המשתמש במבני גרף לשאילתות סמנטיות עם צמתים, קשתות ומאפיינים כדי לייצג ולאחסן נתונים. הגרף מתייחס לפריטי הנתונים באכסון לאוסף צמתים וקשתות, הקשתות מייצגות את היחסים בין הצמתים.
* Redis מאפשר פתרון בעיות מורכבות באמצעות אופטימיזציה של מבנה הנתונים והפקודות המבוצעות באופן מהיר ובפשטות. אף על פי שכל בסיס הנתונים מאוחסן ב־RAM (מתבסס על cache), הוא עדיין מאפשר גיבויים ויציבות. Redis מבוסס על שליפת הנתונים על בסיס "Key-value" המאפשר שליפת נתונים באופן מהיר ביותר מתוך מאגרי מידע ענקיים.

הרצת redis בהתאם לקוד שנשלח בdiscord: (מה שמסומן זה מה שאנחנו רושמים)

תקשורת עם redis באמצעות קונטיינר redis-CLI שמהווה client-

תקשורת עם redis באמצעות python שמהווה client-

**בדיקות Testing:**

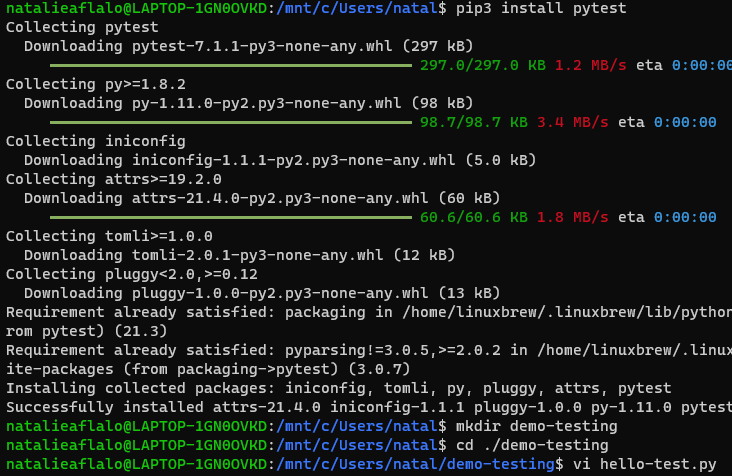
* סוגי בדיקות:

1. Integration test – בודק את המערכת מקצה לקצה. מדמה לקוח שמבצע פעולה כלשהי במערכת ובודק שתוצאת התהליך (שיכול לרוץ בכמה services) תקין.
2. Unit test – בודק service ספציפי באופן לוקלי.

* Profiling- בדיקות זמן לביצוע פעולות במערכת.

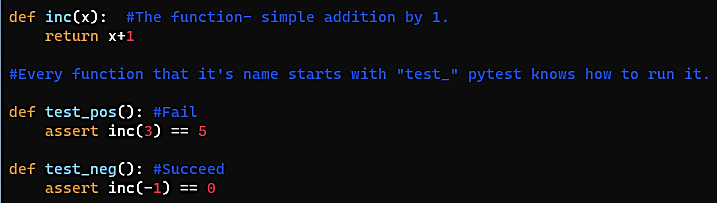
הדגמת ביצוע unit tests באמצעות ספריית pytest:

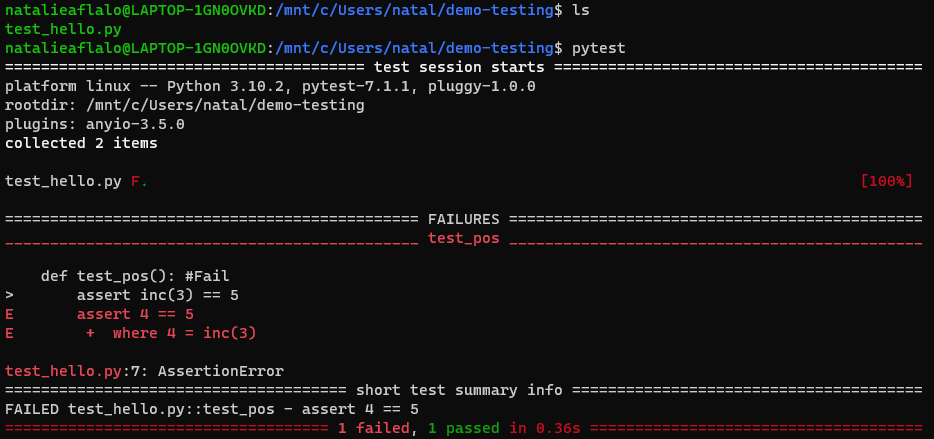
קודם נתקין את pytest . למען הסדר, נעבור לתיקייה אחרת:



ניצור בתיקייה את קובץ הבדיקה- השם שלו חייב להתחיל ב "test\_" כדי ש-pytest יידע להריץ אותו (וגם השמות של פונקציות הבדיקה). בקובץ נבנה פונקציה inc , שאותה נבדוק באמצעות שתי פונקציות נוספות:

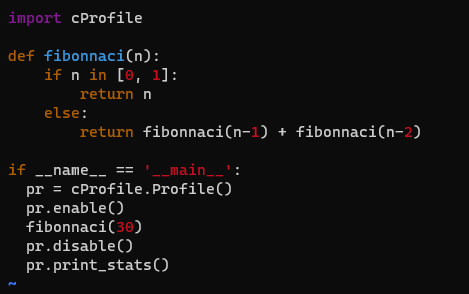
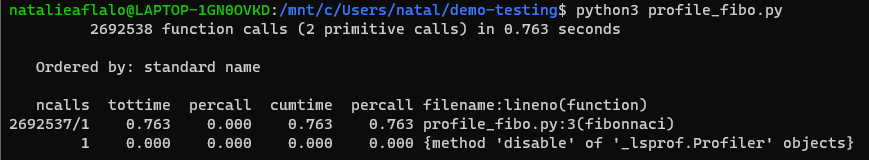




כעת נריץ את pytest בתיקייה עם הקובץ:  


נוכל לראות שטסט אחד נכשל וטסט אחד הצליח כמו שציפינו. בנוסף במקום בו הפונקציה נכשלה רשום מה הוחזר.

הדגמה של profiling:

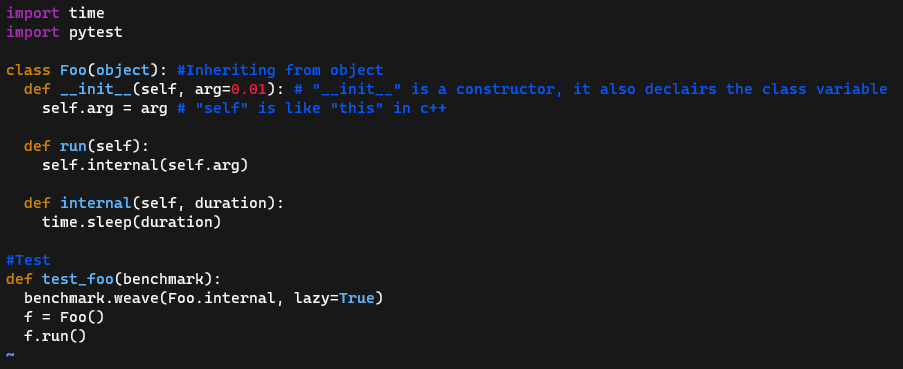
ניצור קובץ פייתון חדש:  
כאן אין צורך בהורדה באמצעות pip, שכן הספרייה cProfile כבר קיימת ומימשנו אותה בקובץ. התנאי if למטה מבצע ריצה לפונקציה ובוחן כמה זמן לקח לה לרוץ. כעת נריץ את הקובץ עצמו באמצעות python:

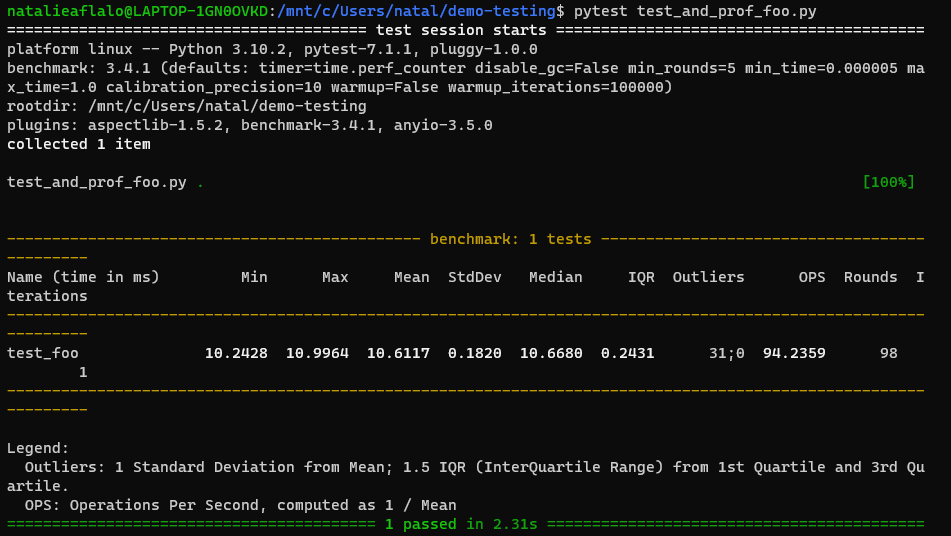
דוגמה נוספת לשימוש ב-pytest עם benchmark:





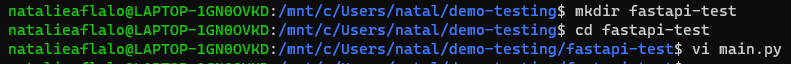


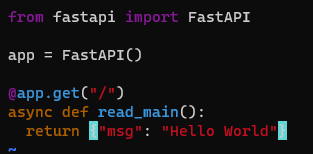




הערה חשובה- עד כה בדוגמאות הבדיקה הייתה באותו קובץ עם מה שהיא בודקת. מדובר ב- bad practice , כיוון שמקובל להפריד בינהם. בדוגמה הבאה נפריד בין קובץ main לקובץ הבודק אותו ונראה כיצד הם מתקשרים.

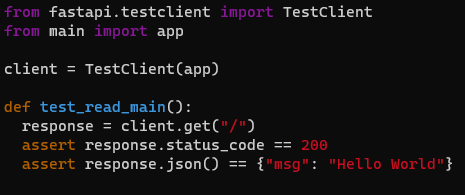
דוגמה לשימוש ב-pytest עם fastapi:



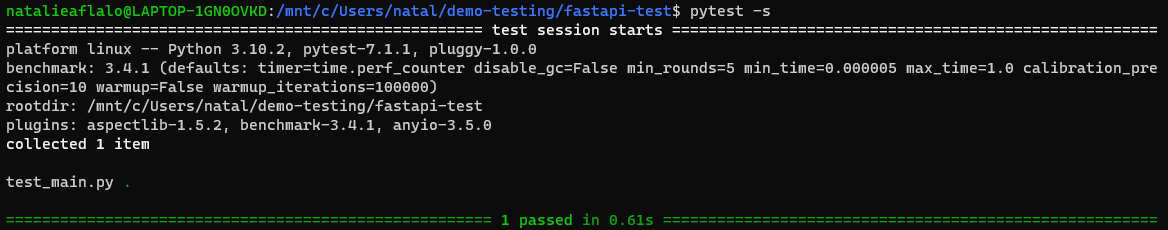




בקובץ הבדיקה נקרא לapp שנמצא בmain

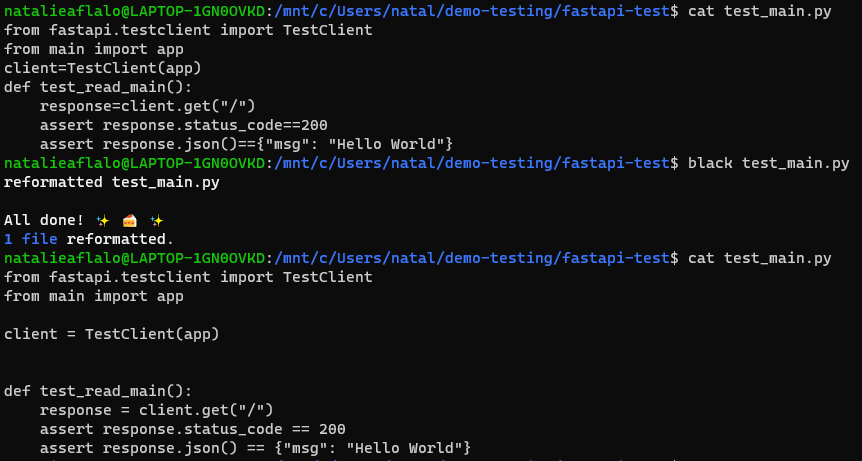


נריץ pytest



**Linting/Formatting using Black:**

Black מבצע סידור נעים לקריאה של קוד. לדוגמה:



ניתן לראות שנוסף ריווח בין השורות ובין סימני = על מנת שהקוד יהיה יותר קריא.

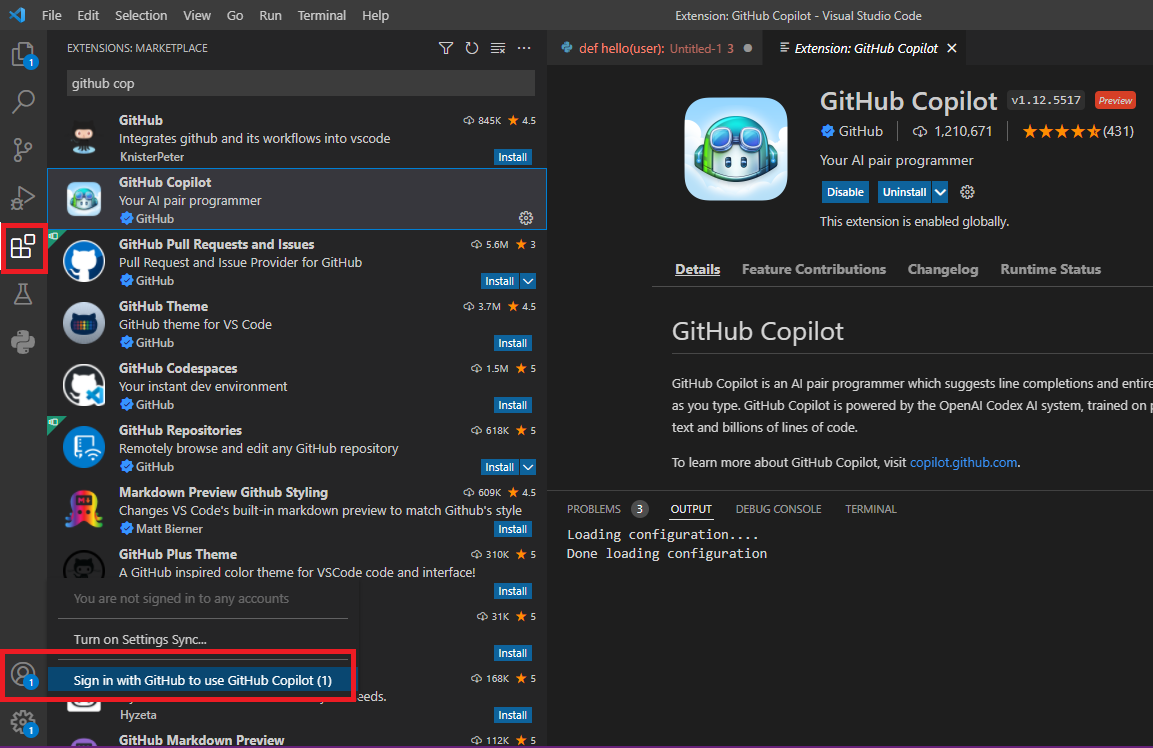
**Async IO:**

תזכורת- עבודה אסינכרונית משמעה לבצע פעולה, ובמקום לחכות עד שהיא תסתיים להמשיך לפעולה הבאה (שלא תלויה בקודמת) ולבאה אחריה. לאורך הזמן מבצעים "דגימות" כדי לראות אם פעולות קודמות החזירו תשובה (הסתיימו) וכך ממשיכים. כלומר, אין תלות בסיום של פעולה כדי להמשיך להתקדם.

Async IO זה עבודה אסינכרונית מול קלט/פלט. לדוגמה- המתנה לתשובה מmicroservice אחר, משרת אחר ברשת (network), מהדיסק, מהזיכרון וכו'.

**Github copilot:**

כלי שנמצא בvisual studio code, מאפשר לבצע חיפוש של קטעי קוד בכל הגיטהאב לפי שמות של פונקציות שנרשום.

תחילה יש להתקין את התוסף ולבצע התחברות לגיטהאב דרך VSCode  


לאחר הסנכרון יש להירשם לרשימת ההמתנה של התוסף הזה, רק לאחר שהם יאשרו יהיה ניתן להשתמש בו.

מסמך README שמדגים כיצד להשתמש בתוסף:   
<https://github.com/github/copilot-docs/blob/main/docs/visualstudiocode/gettingstarted.md#enabling>