פרק הייזום

1. תיאור ראשוני של המערכת

* הפרויקט יהיה מימוש של שירות מקביל ל github שמאפשר ללקוחות מכל העולם להעלות קוד, לשמור אותו ,לשתף אותו, לנהל אותו, לעיין בו, וכו'.
* בחרתי בפרויקט כי אני תומך גדול של open source ויש לי הרבה הערכה לפרויקטים כמו github, gitlab, bitbucket והייתי רוצה בעצמי לממש משהו בסגנון (כמובן גם הפרויקט נראה לי מעניין בגלל הטכנולוגיות המעורבות)

1. הגדרת הלקוח

* המערכת מיועדת למתכנתים בכל העולם ואני מקווה שמתכנתים בכל העולם ישתמשו בה מתכנתים שמעוניינים לשתף קוד וליהנות מקוד שאחרים שיתפו.

1. הגדרת יעדים/מטרות

* שמירה של קוד בענן בעבודה עם git בשביל שמתכנתים יוכלו למקבל עבודה עם אחרים.
* שיתוף של הקוד בשביל להציג אותו למתכנתים אחרים.
* למידה שלי על הדברים האלה והתעסקות איתם

1. בעיות, תועלות וחסכונות

* הבעיות הן שכל אחד יכול להשתמש בgit אבל רק על המחשב שלו והמערכת נועדה לתת פתרון למקבול של עבודה בין מחשבים בעולם.
* המערכת תספק ממשק בשביל לראות מה קורה עם הפרויקט (קבצים נוכחיים, branches קיימים, issues).

1. שירותים שהפרויקט ייתן:

* המערכת תשתלב עם git בשביל שיהיה אפשר לעלות ולהוריד repositories מהמחשב המקומי שלך לשרת בעזרת פקודות כמו (git pull, git push, git clone)
* המערכת תספק ממשק שמציג את הקבצים בתוך repository
* המערכת תספק ממשק שמציג את ה commits שנעשו על כל פרויקט ואת השינויים בכל commit
* המערכת תספק ממשק לעיון ב branchים והצגת המצב של ה repository בכל branch
* המערכת תספק ממשק לעיון ויצירה של issues כדי לעזור לנהל את המשימות של הפרויקט
* המערכת תאפשר למשתמשים לפתוח pull requests על פרויקטים ולבצע code review על השינויים בכל pull request
* המערכת תספק ממשק בשביל עיון וחיפוש של פרויקטים ומשתמשים (כמו רשת חברתית)
* המערכת תספק ממשק בשביל יצירה של פרויקטים

1. הפרויקט מקביל לפרויקטים כמו

* Github: [https://github.com](https://github.com/)
* Gitlab: <https://about.gitlab.com/>
* Bitbucket: <https://bitbucket.org/>

שלושת הפרויקטים הם אותו רעיון כמו הפרויקט שלי רק שבלשלושתם יש CICD

1. האם צפויים קשיים או מגבלות בהגדרת המערכת?

* לא מדובר בכלל בטכנולוגיה חדשנית לא מוכרת מטרת הפרויקט היא למידה אישית שלי.
* דוגמא לסייג זה שהפרויקט לא משתלב עם github

1. תיחום הפרויקט

* הפרויקט מתעסק עם מערכת ההפעלה לינוקס בגלל כל הניהול של השרת git.
* הפרויקט מתעסק ברשתות בגלל התקשורת שצריכה להיות בין השרת ללקוח (מימוש שרת מרובה לקוחות)
* הפרויקט מתעסק עם database בשביל שמירה של כל המידע על המשתמשים.
* הפרויקט מתעסק עם הצפנה בגלל השימוש ב ssh וכל ההודעות בין הלקוח לשרת יהיו מוצפנות בגלל שאנשים לא רוצים שהקוד שלהם בהכרח יהיה פתוח

1. קיימות

* הפרויקט שלי מתקשר לקיימות בגלל שהוא מאפשר פתרון אחר במקום דפים וככה יכול לגרום לאנשים פחות להשתמש ובדפים להשתמש יותר במחשבים.

1. מקרא מושגים:

* Repository

פרויקט של git.

* git pull, git push, git clone

פקודות מובנות בתוך גיט שנועדו בשביל לתאם את הגרסאות המקומיות במחשב של משתמש לגרסאות על השרת

* commits

אוביקט בgit שמגדיר את השינויים שבוצעו בקוד מאז ה commit הקודם

* branch

מערכת בגיט שמאפשרת לפצל עבודה על אותו פרויקט מבלי להפריע למתכנתים האחרים

לכל branch יש היסטוריית commitים משלו (הcommitים שנעשו עד שפתחו את הbranch והcommitים שנעשו עליו) ואפשר לעשות שילוב של branchים

* Github:

מערכת ניהול repositories ופרוייקטים פופלרית

* Issues

פיצ’ר פופולרי במערכות כמו שלי, מאפשר לאנשים להודיע על תקלות בפרויקטים וכך המפתחים יוכלו לדעת ולתקן את הבעיות בצורה נגישה יותר

* pull requests

פיצ’ר פופלרי בפרויקטים כמו שלי, מאפשר למתכנתים של הפרויקט להציג למפתחים שינויים שעשית לבקש מהם לעבור על השינויים שעשית ולהעיר הערות

* CICD

מערכת אוטומטית של העלאה אוטומטית של קוד למשתמשים ומעבר ובדיקות של הקוד בצורה אוטומטית

מסמך אפיון

1. תיאור מערכת:

* המערכת היא מימוש של[git forge](https://en.wikipedia.org/wiki/Forge_(software)) בעצמי שתהיה דומה לgithub, gitlab, bitbucket. המערכת תאפשר ללקוחות מכל העולם להעלות קוד, לשמור אותו ,לשתף אותו, לנהל אותו, לעיין בו, וכו'.

המערכת תכיל שני חלקים:

* שרת git: שרת שמאפשר ללקוח לעשות push, pull, clone, fetch וכו' לrepositories שלהם וככה להתעדכן עם מה שקורה בשרת או לעדכן את השרת מה קרה בשביל משתתפים אחרים בפיתוח הפרויקט
* הרשת החברתית והממשק: הרשת החברתית נועדה לתת ללקוחות אפשרות ליצור משתמש ולאחר מכן: לצפות, לנהל, לשתף, ולשמור הפרויקטים שלהם.

1. תיאור יכולות

* **שרת של git** שמאפשר ללקוחות לעשות pull push clone fetch לrepositories ששמורים על השרת
* **ניהול חשבונות**: אפשרות ליצור חשבון עם מזהה ייחודי ssh key בשביל כל חשבון.
* **הרשאות גישה**: המערכת תאפשר גישה רק אם יש למשתמש הרשאות מתאימים כדי להגן על המידע
* **התחברות:** אפשרות להתחבר לחשבון ולצפות בפרויקטים בהתאם להרשאות שלך
* **זמינות:** המערכת תישאר מחוברת לאחר שהאפליקציה נסגרת כדי להקל על תהליך ההתחברות של המתשמים
* **יצירת פרויקטים דרך הממשק**: אפשרות ליצור פרויקטים של עצמך ולהגדיר לכל פרויקט שם, הרשאות גישה וכו'
* **צפייה בפרויקט**: אפשרות ללקוחות לצפות בקוד של פרויקט בהתאם לcommit ולbranch
* **צפייה ברשימת commits שנעשו:** אפשרות ללקוחות לקבל רשימה של ה commits שנעשו
* **צפייה בשינויים שנעשו בכל commit**: הצגת רשימה של השינויים שנעשו בcommit מסוים
* **יצירת issues**: אפשרות ליצור issues, כלומר: להעיר למפתחים של הפרויקט על תקלה שהמשתמש חווה כדי שהמפתחים יוכלו לתקן את התקלה
* **אפשרות לחיפוש של repositories**
* **יצירת pull requests**: אפשרות ללקוחות ליצור pull request ולהגדיר מאיזה branch ולאיזה branch
* **Code review**: אפשרות למפתח ראשי של פרויקט מסוים לאשר או לפסול pull request בהתאם למה שהוא חושב על השינויים שנעשו לקוד

1. פירוט הבדיקות ('קופסא שחורה')

| **מספר** | **שם הבדיקה (שם שיעיד על תוכן הבדיקה)** | **מה אמורה לבדוק** | **איך מתכננים לבדוק**  **(לתאר בפירוט את שלבי הבדיקה)** |
| --- | --- | --- | --- |
| 1 | בדיקת commit | שלאחר שעשו commit וpush השינויים יוצגו בשרת | לעשות במחשב של הלקוח commit וpush ואז לבקש מהשרת את ה commit הכי חדש ולראות שהכל מעודכן כמו שציפית |
| 2 | בדיקת התחברות | לבדוק שאין אפשרות להתחבר עם סיסמא שגויה ואפשר רק להתחבר עם סיסמא נכונה | לנסות להתחבר למשתמש עם סיסמא שגויה לראות שלא עובד ואז להתחבר עם הסיסמא הנכונה ולראות שעובד |
| 3 | בדיקת branch | שלאחר שיצרו branch השינויים יוצגו בשרת | ליצור במחשב של הלקוח branch ואז לבקש מהשרת את הbranches של פרויקט מסוים ולראות שהכל מעודכן כמו שציפית |
| 4 | בדיקת issue | שמערכת הissues עובדת כמו שצריך | ליצור אצל הלקוח issues והוסיף לissues כמה תגובות. לאחר מכן לבקש מהשרת את הissues ולראות שהכל מעודכן כמו שרצית |
| 5 | בדיקת pull requests | שמערכת ה pull requests עובדת כמו שצריך | ליצור אצל הלקוח pull request ולהוסיף לpull request כמה תגובות. לאחר מכן לבקש מהשרת את הpull request ולראות שהכל מעודכן כמו שרצית |
| 6 | בדיקת בטיחות | בודק שאי אפשר להבין מה כתוב בקבצים על ידי הסנפה כשהם נשלחים ללקוח | לפתוח wireshark, לצפות בהודעה של שליחת הקובץ ולנסות להרכיב את הקובץ שנשלח |
| 7 | עומסים | שמספר משתמשים מתחברים למערכת והכל עובד כנדרש | להתחבר לשרת מכמות גדולה של משתמשים במקביל |
| 8 | יציבות השרת | שלקוחות מתחברים ומתנתקים והשרת נשאר יציב | להתחבר ולהתנק הרבה פעמים במשך הרבה זמן ולראות שהכל עדיין עובד כראוי |
| 9 | אבטחה | שאין ערבוב מידע בין repositories של אנשים שונים | לעשות clone לrepository ולראות שכל בקבצים שאתה מצפה נמצאים ושאין משהו שלא אמור להיות שם |

1. תכנון לוח זמנים

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **פעילות** | **זמן התחלה מתוכנן** | **זמן סיום מתוכנן** | **זמן התחלה בפועל** | **זמן סיום בפועל** | **הערות** |
| יזום | 1.11.23 | 23.11.23 | 4.11.23 | 8.11.23 | הוגש לפני הזמן |
| אפיון | 23.11.23 | 1.12.23 | 9.11.23 | 7.12.23 |  |
| ניתוח | 1.12.23 | 15.12.23 | 7.12.23 | 15.12.24 |  |
| עיצוב | 15.12.23 | 15.1.24 | 15.12.24 | 29.1.24 | הגשה ראשונה הייתה בזמן אבל בגלל שחליתי בשפעת התיקון הוגש אחרי הזמן |
| התקנה והתחלה של שרת ה git | 15.1.24 | 25.1.24 | 14.1.24 | 14.1.24 |  |
| הוספת database | 25.1.24 | 28.1.24 | 14.1.24 | 29.1.24 |  |
| ניהול שירות חשבונות | 28.1.24 | 5.2.24 |  |  |  |
| שילוב עם שרת ה git והנגשת קבצים | 5.2.24 | 12.2.24 |  |  |  |
| מימוש issues ו pull requests | 12.2.24 | 25.2.24 |  |  |  |
| גרסה ראשונית | 25.2.24 | 1.3.24 |  |  |  |
| מסמך בדיקות | 1.3.24 | 10.4.24 |  |  |  |
| מדריך למשתמש | 10.4.24 | 1.5.24 |  |  |  |
| גרסה סופית | 10.5.24 | 10.5.24 |  |  |  |
| סגירת תיק פרויקט |  |  |  |  |  |
| מצגת הסבר |  |  |  |  |  |

1. ניהול סיכונים בפרויקט

| **הסיכון** | **פירוט הסיכון** | **רמת הסיכון**  **(קל/ בינוני/ קשה)** | **תיאור דרכים (לפחות 2 ) להתמודדות עם הסיכון ולהקטין אותו** | **מה בוצע בפועל** | **תאריך** |
| --- | --- | --- | --- | --- | --- |
| אי עמידה בזמנים | פרויקט לא יושלם | קשה | * הקדמת לו"זים משימות * ארגון הזמן בצורה יעילה * להתחיל בדברים היותר קשים | עמדתי בזמנים |  |
| כישלון בסידור שרת ה git | לא יהיה שום דבר בפרויקט כי הכל בנוי על זה | קשה | * לקרוא חומרים על הנושא * להתחיל מוקדם במציאת פתרון * למצוא פתרון פשוט יותר ובסיסי יותר | הצלחתי להתממשק עם שרת הgit |  |
| לא תהיה דרך נוחה לתקשר עם git ו python | זה יקשה על תהליך בניית הפרויקט | קל | * לקרוא חומרים על הנושא * להתחיל מוקדם במציאת פתרון * בניית תקשורת כזאת בעצמי * הרצת תהליכים שיריצו פשוט פקודות כמו בterminal | הייתה דרך נוחה |  |
| ממשק משתמש לא נוח | המשתמש לא יצליח לעשות את הדברים שהוא רוצה בצורה קלה | בינוני | * עיצוב מחדש של הממשק ומימוש שלו * יצירת cli במקום ממשק גרפי   (cli: ממש שורת פקודה) | הצליח לעשות את הדברים שרציתי בצורה נוחה |  |
| זמני תגובה איטיים | המשתמש לא ירצה להשתמש באפליקציה בגלל שהכל איטי | קל ->  קשה -> | * הרצה במקביל * ניתוח זמנים ושינוי של המימוש בשביל שיהיה יותר מהיר | לא היו |  |
| נאבד ה database | אצטרך לעשות הרבה עבודה שוב | בינוני | * לגבות את ה database * לשמור הסבר של בנייה מחדש | לא קרה |  |
| נהרס המחשב שלי שירוץ עליו השרת | אצטרך להתחיל מחדש את ה database ושרת git | בינוני | * לעבוד עם containers ולגבות את מצב המחשב | לא קרה |  |

מסמך ניתוח

שם היכולת: **אפשרות התחברות לשרת בעזרת לקוח של ה git cli**

מהות היכולת: אפשרות בשביל לקוחות לעשות pull push fetch clone במחשבים שלהם כשהם מתחברים לשרת

**צד שרת:**

אוסף יכולות:

* + עבודה עם מערכת הקבצים
  + עבודה עם שרת לינוקס
  + ניהול ssh keys
  + ssh
  + התקנות מתאימות בלינוקס
  + תקשורת
  + התנקת מערכת בעזרת dockerfile

רשימת אובייקטים: linux, בסיס נתונים, תקשורת

**שרת:**

שם היכולת: **קבלת תשובה מהשרת**

מהות היכולת: החזרת תשובה לפי פרוטוקול ומוצפנת מהשרת ללקוח

**צד שרת:**

אוסף יכולות:

* + בניית תשובה לפי הפרוטוקול
  + הצפנה
  + שליחה

רשימת אובייקטים: פרוטוקול, הצפנה, תקשורת

לקוח:

אוסף יכולות:

* + קבלת תשובה
  + פענוח
  + פירוק לפי פרוטוקול
  + הצגה למשתמש
* רשימת אובייקטים: פרוטוקול, פענוח, תקשורת, ממשק משתמש

**לקוח:**

שם היכולת: **שליחת בקשה לשרת**

מהות היכולת: שליחת בקשה לפי פרוטוקול ומוצפנת מהלקוח לשרת

**צד לקוח:**

אוסף יכולות:

* + בניית בקשה לפי הפרוטוקול
  + הצפנה
  + שליחה

רשימת אובייקטים: פרוטוקול, הצפנה, תקשורת

שרת:

אוסף יכולות:

* + קבלת בקשה
  + פענוח
  + פירוק לפי פרוטוקול

רשימת אובייקטים: פרוטוקול, פענוח, תקשורת, ממשק משתמש

שם היכולת: **הרשמה למערכת**

מהות היכולת: רישום משתמש חדש במערכת

**צד לקוח:**

אוסף יכולות:

* + ממשק משתמש – מסך הרשמה
  + קליטת נתונים
  + בדיקת תקינות
  + **שליחת בקשה לשרת**
  + **קבלת תשובה מהשרת**
  + כתיבת קובץ

רשימת אובייקטים: ממשק משתמש, מפתח התחברות, מערכת קבצים

**צד שרת:**

אוסף יכולות:

* + **שליחת בקשה לשרת – צד שרת**
  + בדיקה מול בסיס הנתונים
  + גיבוב סיסמא
  + הוספה לבסיס נתונים
  + יצירת, מפתח התחברות
  + **קבלת תשובה מהשרת – צד שרת**

רשימת אובייקטים: בסיס נתונים, מערכת קבצים

שם היכולת: **התחברות**

מהות היכולת: להתחבר לחשבון קיים

**צד לקוח:**

אוסף יכולות:

* + ממשק משתמש – מסך התחברות
  + **שליחת בקשה לשרת**
  + **קבלת תשובה מהשרת**
  + התחברות אוטומטית

רשימת אובייקטים: ממשק משתמש, מערכת קבצים

**צד שרת:**

אוסף יכולות:

* + **שליחת בקשה לשרת – צד שרת**
  + בדיקה מול בסיס הנתונים
  + **קבלת תשובה מהשרת – צד שרת**

רשימת אובייקטים: בסיס נתונים

שם היכולת: **יצירת פרויקט**

מהות היכולת: לאפשר ללקוח ליצור פרויקט חדש

**צד לקוח:**

אוסף יכולות:

* + **התחברות**
  + ממשק משתמש – יצירת פרויקט
  + קליטת נתונים
  + בדיקת תקינות
  + **שליחת בקשה לשרת**
  + **קבלת תשובה מהשרת**

רשימת אובייקטים: ממשק משתמש

**צד שרת:**

אוסף יכולות:

* + **שליחת בקשה לשרת – צד שרת**
  + בדיקת חשבון מול בסיס הנתונים
  + הוספה לבסיס נתונים
  + יצירת הפרויקט בתיקייה ריקה במערכת הקבצים
  + **קבלת תשובה מהשרת – צד שרת**

רשימת אובייקטים: בסיס נתונים, מערכת קבצים~~,~~ git

שם היכולת: **צפייה בקובץ בפרויקט**

מהות היכולת: לאפשר ללקוח לצפות בפרויקט (אם יש לו הרשאות מתאימות)

**צד לקוח:**

אוסף יכולות:

* **התחברות**
* **חיפוש repositories**
* ממשק משתמש – מסך צפייה בפרויקט
* ממשק משתמש – מסך צפייה בקובץ
* הצגת תשובה מהשרת בממשק
* פריסת קבצים בקבלה
* **שליחת בקשה לשרת**
* **קבלת תשובה מהשרת**

רשימת אובייקטים: ממשק משתמש

**צד שרת:**

אוסף יכולות:

* + בדיקת הרשאות חשבון מול בסיס הנתונים
  + דחיסת קבצים בשליחה
* **שליחת בקשה לשרת – צד שרת**
* **קבלת תשובה מהשרת – צד שרת**

רשימת אובייקטים: בסיס נתונים, מערכת קבצים

שם היכולת: **צפייה בפרויקט**

מהות היכולת: לאפשר ללקוח לצפות בפרויקט (אם יש לו הרשאות מתאימות)

**צד לקוח:**

אוסף יכולות:

* **התחברות**
* **חיפוש repositories**
* ממשק משתמש – מסך צפייה בפרויקט
* הצגת תשובה מהשרת בממשק
* **שליחת בקשה לשרת**
* **קבלת תשובה מהשרת**

רשימת אובייקטים: ממשק משתמש

**צד שרת:**

אוסף יכולות:

* + בדיקת הרשאות חשבון מול בסיס הנתונים
  + דחיסת קבצים בשליחה
* **שליחת בקשה לשרת – צד שרת**
* **קבלת תשובה מהשרת – צד שרת**

רשימת אובייקטים: בסיס נתונים, מערכת קבצים

שם היכולת: **צפייה ברשימה של ה commits שנעשו**

מהות היכולת: לאפשר ללקוח לראות איזה commits נעשו

**צד לקוח:**

אוסף יכולות:

* **התחברות**
* **חיפוש repositories**
* ממשק משתמש – מסך צפייה בcommits
* הצגת תשובה מהשרת בממשק
* **שליחת בקשה לשרת**
* **קבלת תשובה מהשרת**

רשימת אובייקטים: ממשק משתמש

**צד שרת:**

אוסף יכולות:

* + בדיקת הרשאות חשבון מול בסיס הנתונים
  + דחיסת קבצים בשליחה
* **שליחת בקשה לשרת – צד שרת**
* **קבלת תשובה מהשרת – צד שרת**

רשימת אובייקטים: בסיס נתונים, מערכת קבצים

שם היכולת: **צפייה בשינויים שנעשו ב commit מסוים**

מהות היכולת: לאפשר ללקוח לצפות ולהבין שינוים שנעשו בפרויקט

**צד לקוח:**

אוסף יכולות:

* **התחברות**
* ממשק משתמש – מסך צפייה בשינויי commit
* ממשק משתמש – אפשרות לבחירת commit שבו רוצים לצפות
* הצגת תשובה מהשרת בממשק
* **שליחת בקשה לשרת**
* **קבלת תשובה מהשרת**

רשימת אובייקטים: ממשק משתמש

**צד שרת:**

אוסף יכולות:

* + בדיקת הרשאות חשבון מול בסיס הנתונים
  + דחיסת קבצים בשליחה
* **שליחת בקשה לשרת – צד שרת**
* **קבלת תשובה מהשרת – צד שרת**

רשימת אובייקטים: בסיס נתונים, מערכת קבצים, git

שם היכולת: **יצירת issue**

מהות היכולת: אפשרות ללקוחות ליצור issue על פרויקט מסוים

**צד לקוח:**

אוסף יכולות:

* **התחברות**
* ממשק משתמש – ממשק ליצירת issue על פרויקט מסוים
* הצגת הצלחת יצירה ללקוח
* **שליחת בקשה לשרת**
* **קבלת תשובה מהשרת**

רשימת אובייקטים: ממשק משתמש

**צד שרת:**

אוסף יכולות:

* + הוספה לבסיס נתונים
  + בדיקת הרשאות חשבון מול בסיס הנתונים
* **שליחת בקשה לשרת – צד שרת**
* **קבלת תשובה מהשרת - צד שרת**

רשימת אובייקטים: בסיס נתונים

שם היכולת**: חיפוש repositories**

מהות היכולת: אפשרות ללקוחות לחפש פרויקטים לפי שם

**צד לקוח:**

אוסף יכולות:

* **התחברות**
  + ממשק משתמש – אפשרות לחיפוש פרויקט
  + הצגת הצלחת יצירה ללקוח
* **שליחת בקשה לשרת**
* **קבלת תשובה מהשרת**

רשימת אובייקטים: ממשק משתמש

**צד שרת:**

אוסף יכולות:

* + בדיקת הרשאות חשבון מול בסיס הנתונים
* **שליחת בקשה לשרת – צד שרת**
* **קבלת תשובה מהשרת - צד שרת**

רשימת אובייקטים: בסיס נתונים, מערכת קבצים?

שם היכולת: **יצירת pull request**

מהות היכולת: אפשרות ללקוחות ליצור pull request ולהגדיר מאיזה branch ולאיזה branch

**צד לקוח:**

אוסף יכולות:

* **התחברות**
  + ממשק משתמש – יצירת pull request
  + הצגת הצלחת יצירה ללקוח
* **שליחת בקשה לשרת**
* **קבלת תשובה מהשרת**

רשימת אובייקטים: ממשק משתמש

**צד שרת:**

אוסף יכולות:

* + בדיקת הרשאות חשבון מול בסיס הנתונים
  + תקשורת עם git בשביל בדיקת branches
* **שליחת בקשה לשרת – צד שרת**
* **קבלת תשובה מהשרת - צד שרת**

רשימת אובייקטים: בסיס נתונים, מערכת קבצים, git

שם היכולת: **Code review**

מהות היכולת: לאפשר למנהלים של פרויקט מסוים לאשר pull request

**צד לקוח:**

אוסף יכולות:

* **התחברות**
  + ממשק משתמש – ממשק code review
  + הצגת שינויים שנעשו
* **שליחת בקשה לשרת**
* **קבלת תשובה מהשרת**

רשימת אובייקטים: ממשק משתמש

**צד שרת:**

אוסף יכולות:

* + בדיקת הרשאות חשבון מול בסיס הנתונים
  + דחיסה של קובץ המכילים שינויים
* **שליחת בקשה לשרת – צד שרת**
* **קבלת תשובה מהשרת – צד שרת**

רשימת אובייקטים: בסיס נתונים

שם היכולת: **לכתוב comments על issues וpull requests**

מהות היכולת: לאפשר תקשורת בין המפתחים של האפליקציה

**צד לקוח:**

אוסף יכולות:

* **התחברות**
  + ממשק משתמש – ממשק לכתיבת comment על issues/pull request
* **שליחת בקשה לשרת**
* **קבלת תשובה מהשרת**

רשימת אובייקטים: תקשורת, הצפנה/פענוח, ממשק משתמש

**צד שרת:**

אוסף יכולות:

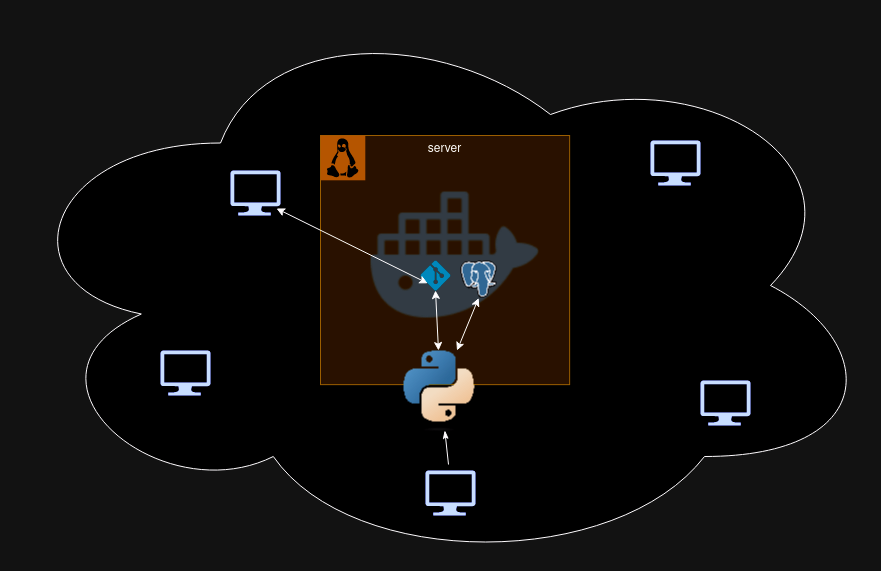
* + הוספה לבסיס נתונים
  + בדיקת הרשאות חשבון מול בסיס הנתונים
* **שליחת בקשה לשרת – צד שרת**
* **קבלת תשובה מהשרת – צד שרת**

רשימת אובייקטים: בסיס נתונים, תקשורת, פענוח/הצפנה

מסמך עיצוב

תיאור הארכיטקטורה של המערכת המוצעת:

* הריבוע הכתום באמצע הוא מחשב השרת של המערכת (מחשב לינוקס).
* השרת מריץ docker עם שני containers והם - gitolite וpostgres ואת הקוד של השרת שכתוב בpython
* postgres זה הdatabase ורק הpython מתקשר איתו
* gitolite אחראי לנהל את כל מה שקשור לgit (לאחסן את הפרוייקטים, לאפשר להוריד אותם עם git clone, לדחוף שינויים עם git push) גם הלקוחות יכולים לתקשר איתו (אבל לא יכולים לעשות הכל בגלל ההרשאות שלהם) וגם הpython מתקשר איתו כמנהל
* לקוחות יכולים לתקשר עם הpython וכך מבצעים את רוב היכולות של הפרויקט

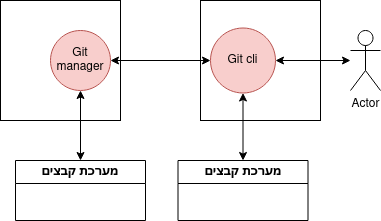


תיאור הטכנולוגיה הרלוונטית:

* שפת תכנות
  + python – שפת הפיתוח העיקרית
  + sql – שפת השאילתות לבסיס הנתונים שאשתמש לפרויקט
  + bash – שפת התקשורת עם מערכת ההפעלה linux בשרת
* מערכת בפעלה
  + שרת – Linux
  + לקוח – מחשב שמסוגל להריץ python
  + בגלל שהפרוייקט כתוב בpython אמורה להיות אפשרות להרצה של פרויקט בכל מכשיר שמסוגל להריץ python
* תקשורת
  + TCP.
  + בגלל שהתקשורת שלי בפרויקט נעשית על ידי הודעות יחידות וברורות בין הלקוח והשרת אי אפשר שיהיו מצבים של קבלת מידע לא תקין, ואין בעיה שאולי ייקח להודעה יותר זמן להגיע
  + שרת מרובה לקוחות
* תחומי עניין
  + הצפנה – סימטרית
  + multi threading
  + דחיסה ופריסה של קבצים בשליחה
  + מערכת הפעלה Linux

תיאור מודולים בהם נעשה שימוש:

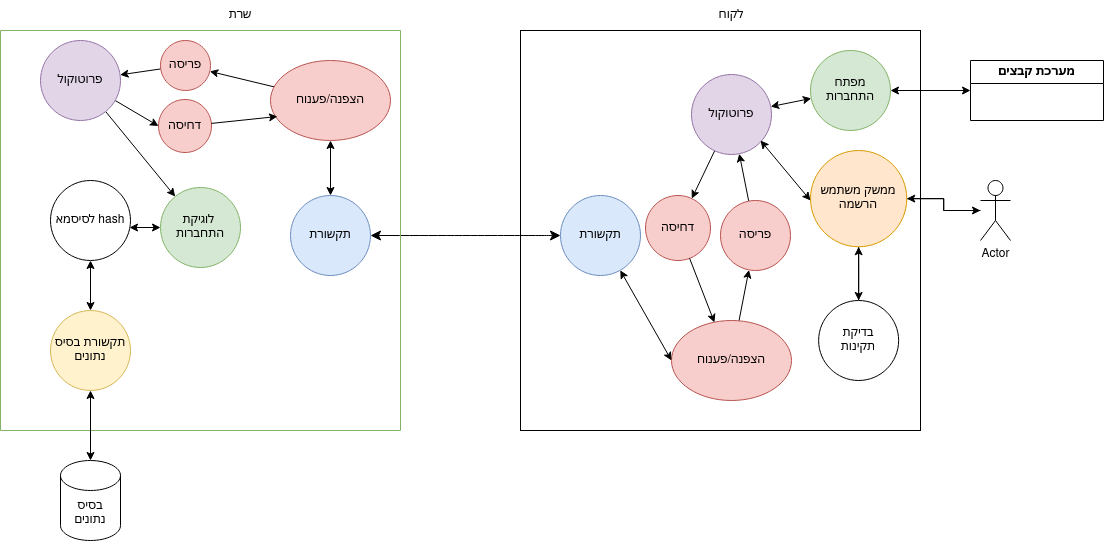
שרת Git:



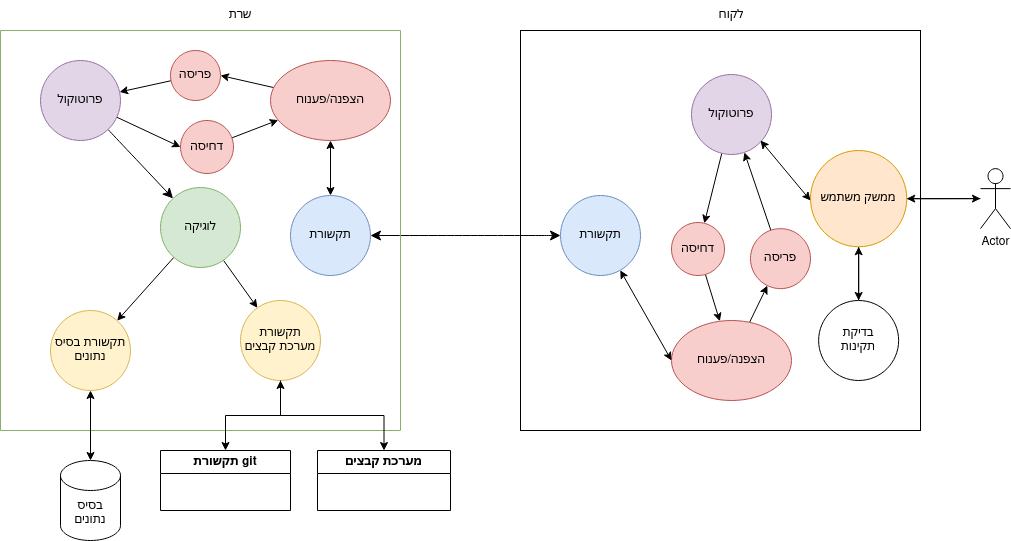
הרשמה:



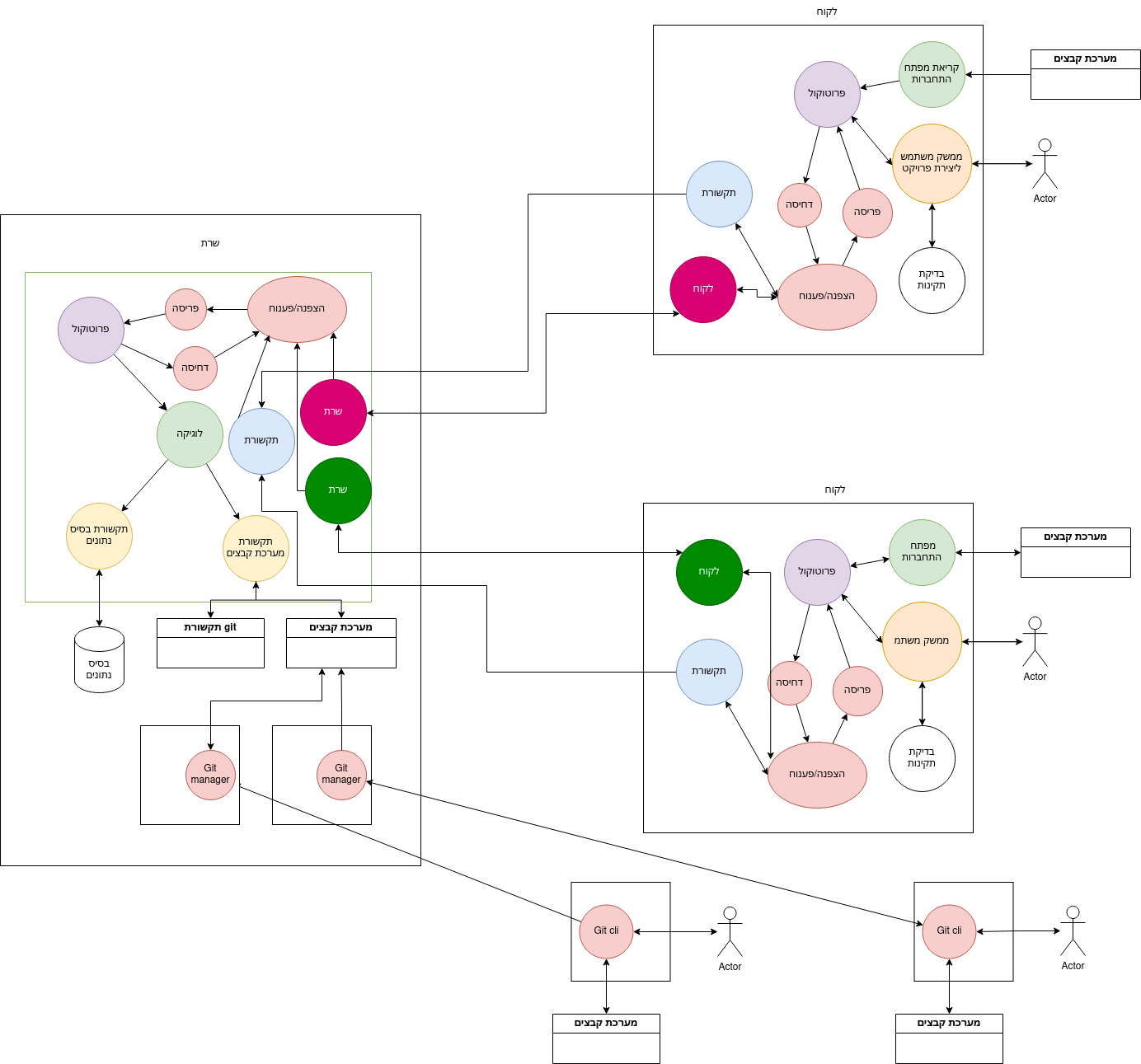
התחברות:



יצירת פרויקט, צפייה בפרויקט, יצירת pull request, צפייה בשינויים שנעשו ב commit, חיפוש repositories, צפייה בקובץ בפרויקט, יצירת issue, לכתוב comments על issues ו pull requests



דיאגרמה מלאה - שרת מרובה לקוחות:



תיאור סביבת הפיתוח:

שפת הפיתוח: Python

כלי הפיתוח:

* Python 3.10 – הגרסא הכי עדכנית של python בתקופת פיתוח הפרויקט
* Pycharm – לכתיבת הקוד ו Debugging של הקוד
* Docker – להתממשקות עם ה container
* ssh – להתחברות עם השרת מרחוק
* git – לשימוש ביכולת העיקרית של הפרוייקט
* vim – לעריכת הקבצים של שרת הlinux
* file explorer – לצפייה בקבצי הפרויקט
* wireshark – לצפייה במה שעובר ברשת

סוגיות בפרוייקט:

האם להריץ את השרת על linux או windows?

|  |  |
| --- | --- |
| Linux | Windows |
| יתרונות   * נתמך על ידי Git * הרבה כלי עזר בשביל שרת Git * מעניין אותי אישית * יותר נפוץ כ container אז יהיה יותר דוגמאות באינטרנט   חסרונות   * יותר קשה כסביבת עבודה במעבדה שבה כל המחשבים הם windows | חסרונות   * לא נתמך על ידי Git * בגלל שלא נתמך לא קיימים כל כך הרבה כלי עזר * פחות מעניין אותי אישית   יתרונות   * נמצא במעבדה אז יותר קל לייצר סביבת עבודה |

**בחרתי בLinux, היתרונות בבירור מנצחות את החסרונות.**

האם להשאיר את socket פתוח לטיפול בכלל בקשות מלקוח מסוים? או כל פעם ליצור סוקט חדש בשביל כל בקשה, מתוך כוונה לחקות את פרוטוקול HTTP שכך עובדים מול ה git : request-response?

|  |  |
| --- | --- |
| ליצור חדש | להשאיר פתוח |
| יתרונות   * מפשט את הקוד כי לא צריך לדאוג שהסוקט חי ולהתעסק בחיבורים ניתוקים * בגלל שמשתמשים באפליקייה בצורה שבקשות לא בהכרח נעשות בצורה קבועה ויכול לעבור פרק זמן ארוך בין כל בקשה, להשאיר סוקט פתוח זה בזבוז של משאבים   חסרונות   * דורש התחברות על כל בקשה * דורש מהלקוח להשיג מפתח הצפנה חדש על כל בקשה | חסרונות   * מסבך את הקוד יותר כי צריך לדאוג שהחיבור חי תמיד * יכול להיווצר מצב של הרבה סוקטים שלא עוברת בהן שום תקשורת לאורך זמן   יתרונות   * הsession זוכר את התחברות המשתמש * לא דורש השגת מפתח הצפנה כל פעם |

**בחרתי ליצור סוקט חדש כל פעם. היתרונות בצורת מימוש זאת מתאימה יותר לפרוייקט שלי.**

האם להעביר הודעות כjson או בפורמט שאמציא?

| JSON | פורמט ממוצא |
| --- | --- |
| יתרונות   * תמיכה מאוד רחבה על ידי הרבה ספריות * קריא מאוד * ורסטילי מאוד * מקבע את כל ההודעות למבנה מסוים * יש בו הרבה אותיות בלי משמעות אז נוצר קובץ גדול – אפשר לפתור את זה על ידי דחיסה   חסרונות   * אני צריך ללמוד איך לעבוד עם קובץ כזה | חסרונות   * אני צריך לממש את ה parsing וencoding * אני ממציא את הגלגל מחדש   יתרונות   * מאפשר לי חופשיות מוחלטת לעצב איך שאני רוצה |

**בחרתי להשתמש בjson בגלל שהיתרונות של json מאוד משמעותיות והחסרונות של פורמט עצמאי מאוד משמעותיות**

מימוש תקשורת

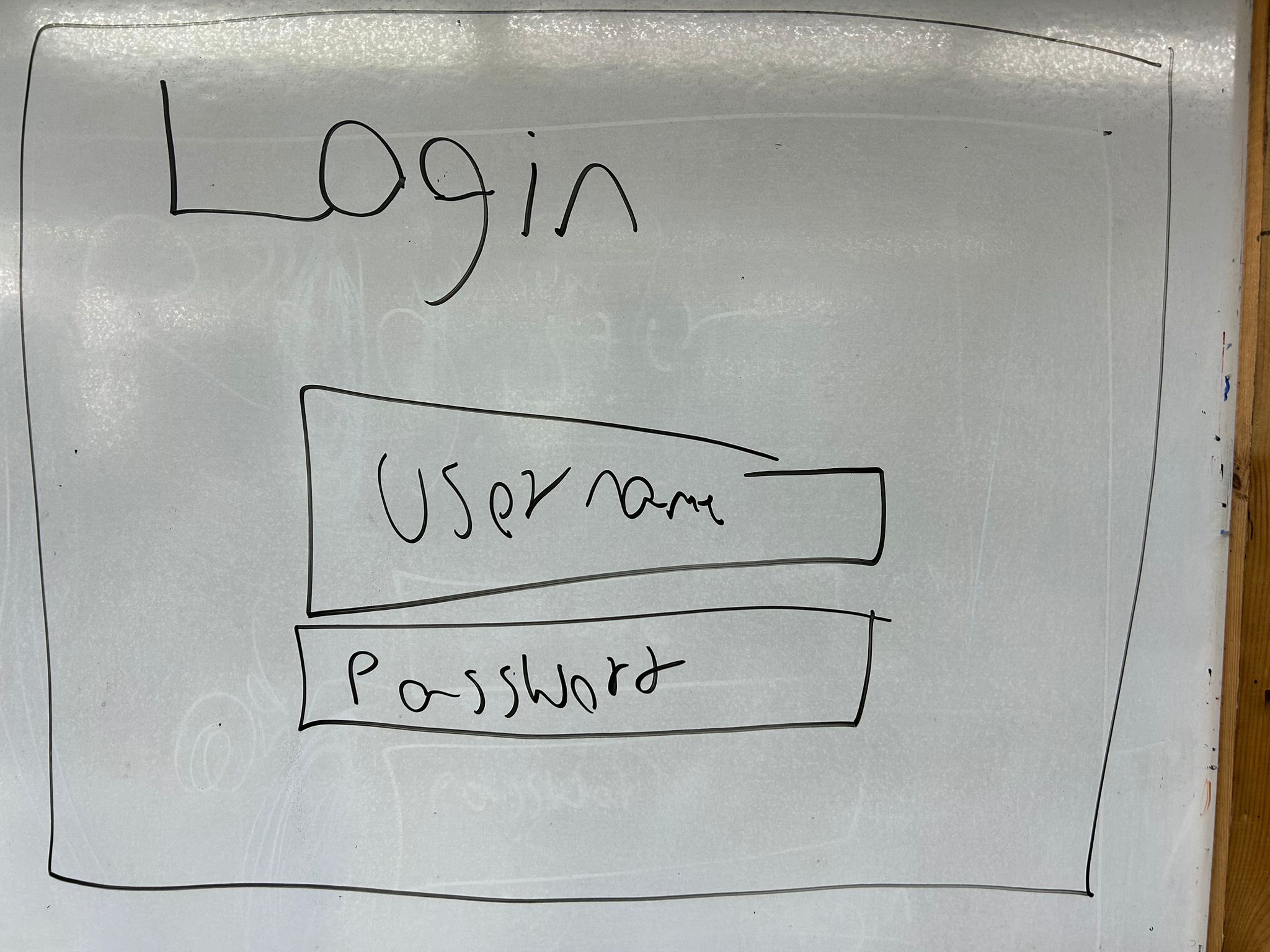
|  |  |  |
| --- | --- | --- |
| סוקט יחיד  כל לקוח מתחבר לאותו סוקט עבור הצרכים | שני סוקטים נפרדים -  סוקט לדברים כללים וסוקט נוסף לטיפול בקבצים | הרבה סוקטים –  סוקט לדברים כללים וסוקט פרטי עבור כל לקוח לטיפול בקבצים |
| יתרון   * מימוש פשוט   חסרון   * יוצר עומס גדול על הסוקט שמוביל לעיכובים * דורש בפרדה בין קובץ להודעה כללית | יתרון   * מימוש פשוט * קיימת הפרדה ברורה בין קבצים הודעה כללית   חסרון   * עדיין קיים עומס על הסוקט של הקבצים ויכול להיווצר פקק | יתרון   * הפרדה מוחלטת בין המידע * הפרדה מוחלטת בין הלקוחות ואין עיכובים   חסררון   * מורכב יותר למימוש |

**אני בוחר בפתרון השלישי בגלל שאני אוהב את ההפרדה והסידור ואין לי בעיה עם פתרון מורכב יותר**

תיאור מסכים

**מסך login**

מאפשר התחברות ומעבר למסך הראשי או למעבר למסך sign up

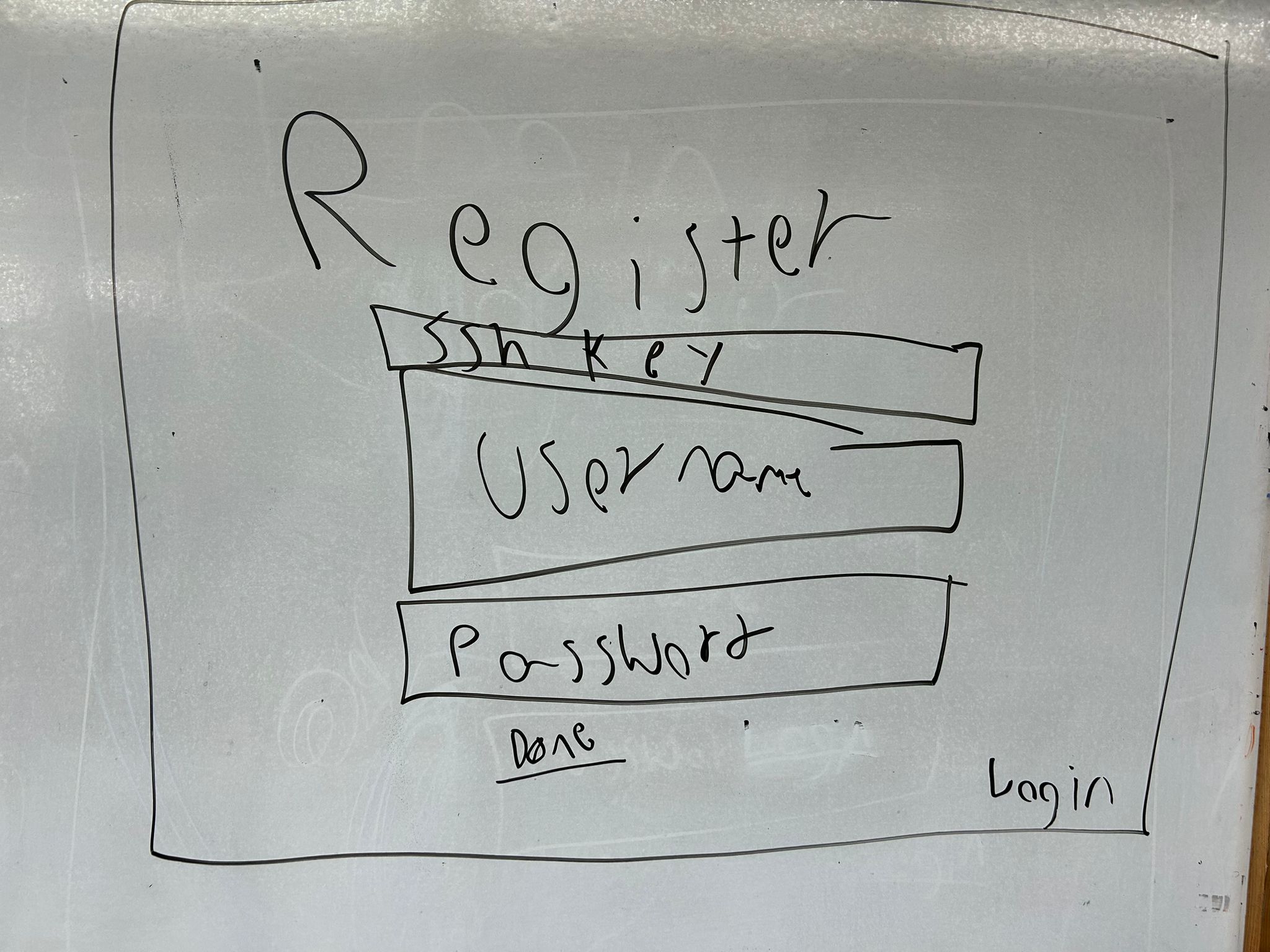


Sign up

**מסך register**

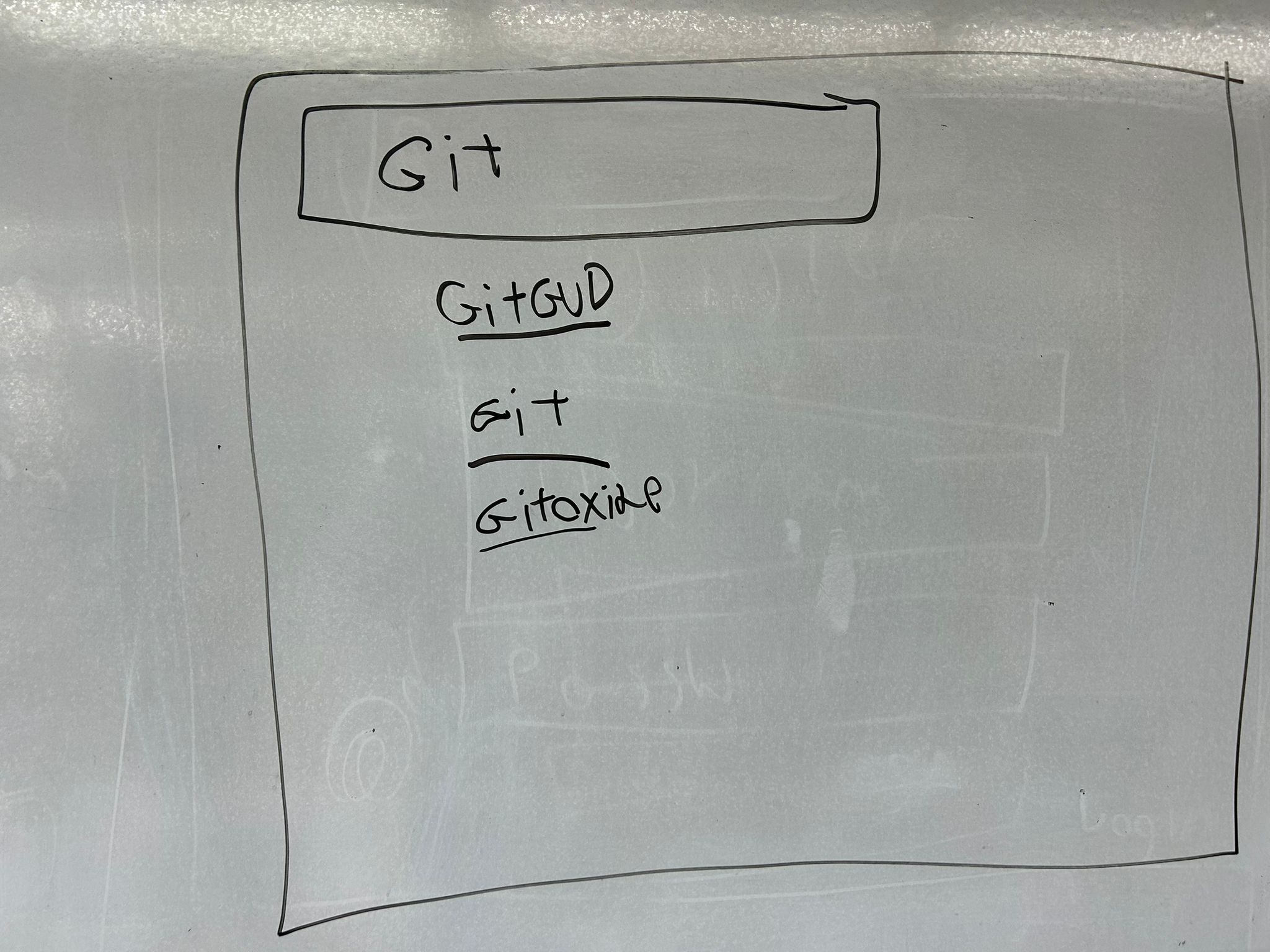
מאפשר מעבר למסך ראשי ולמסך login

דורש מהמשתמש שם משתמש סיסמא וssh-key



**מסך חיפוש repository**

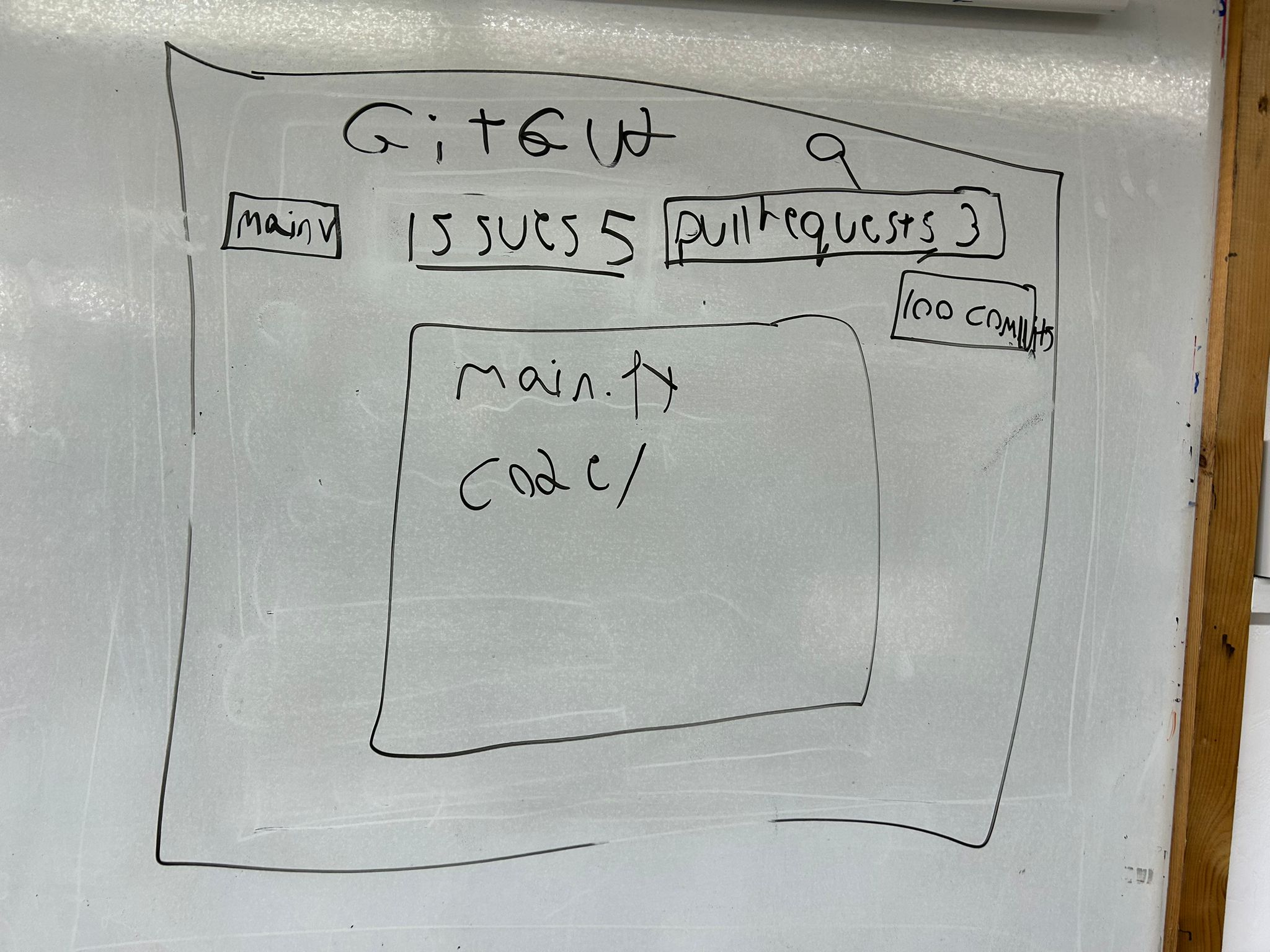
מאפשר חיפוש לפי שם מאפשר מעבר למסך repository



**מסך ראשי של repository**

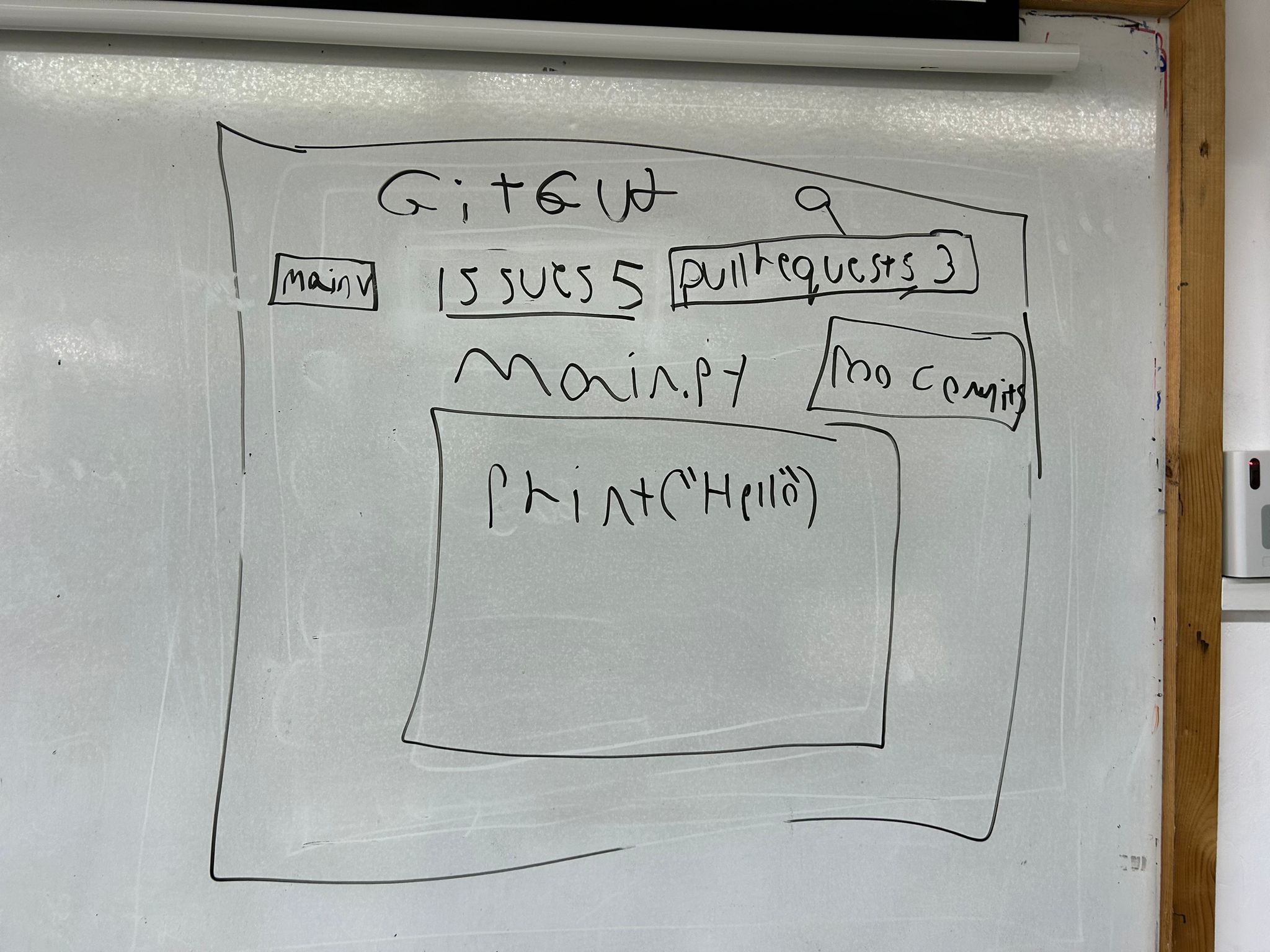
מאפשר מעבר לissues, pull requests, קבצים בקוד תיקיות שונות בקבצים ו commits

המסך מאפשר גם מעבר בין branches



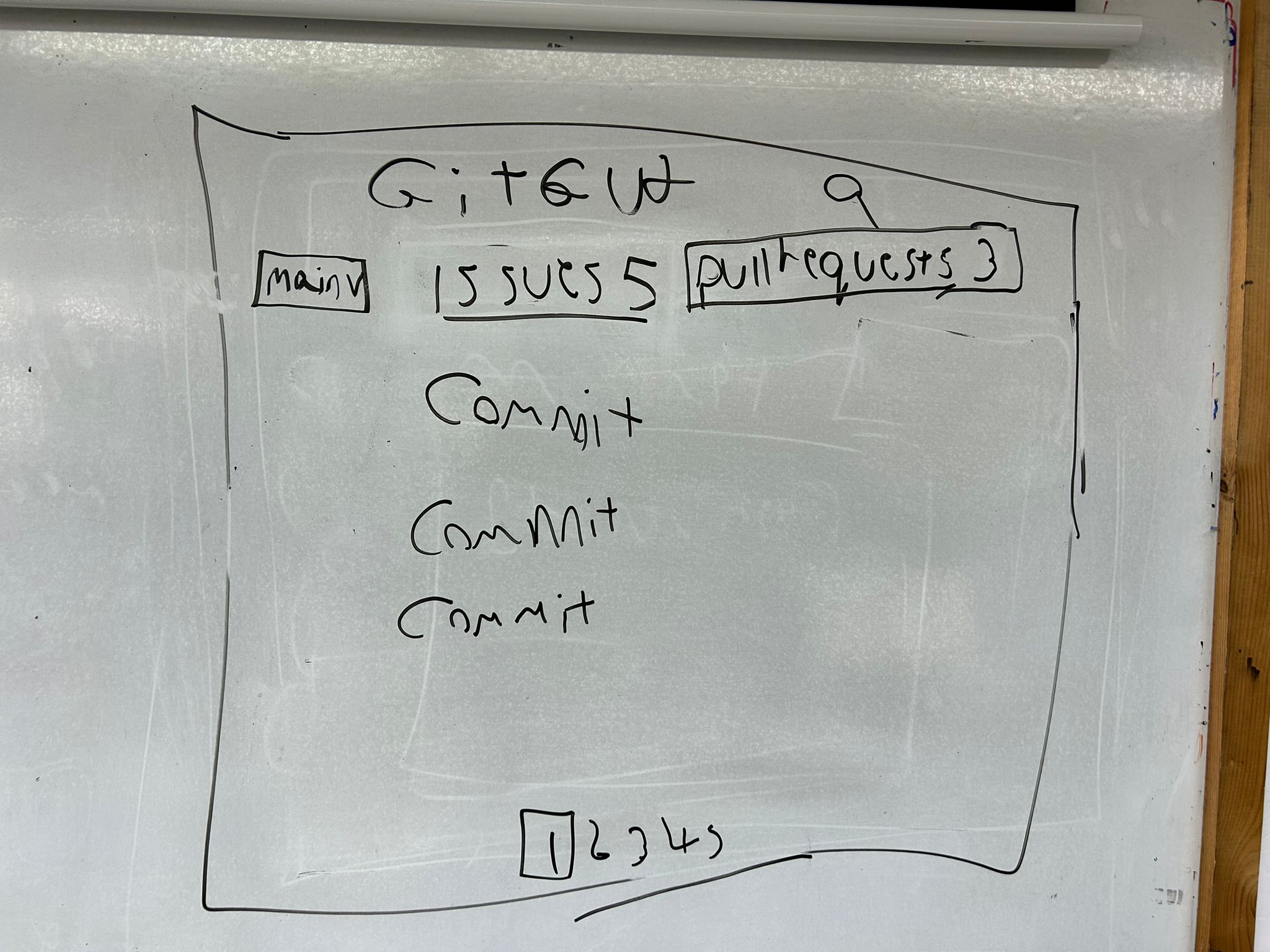
**מסך צפייה בקובץ**

מציג את התוכן של קובץ



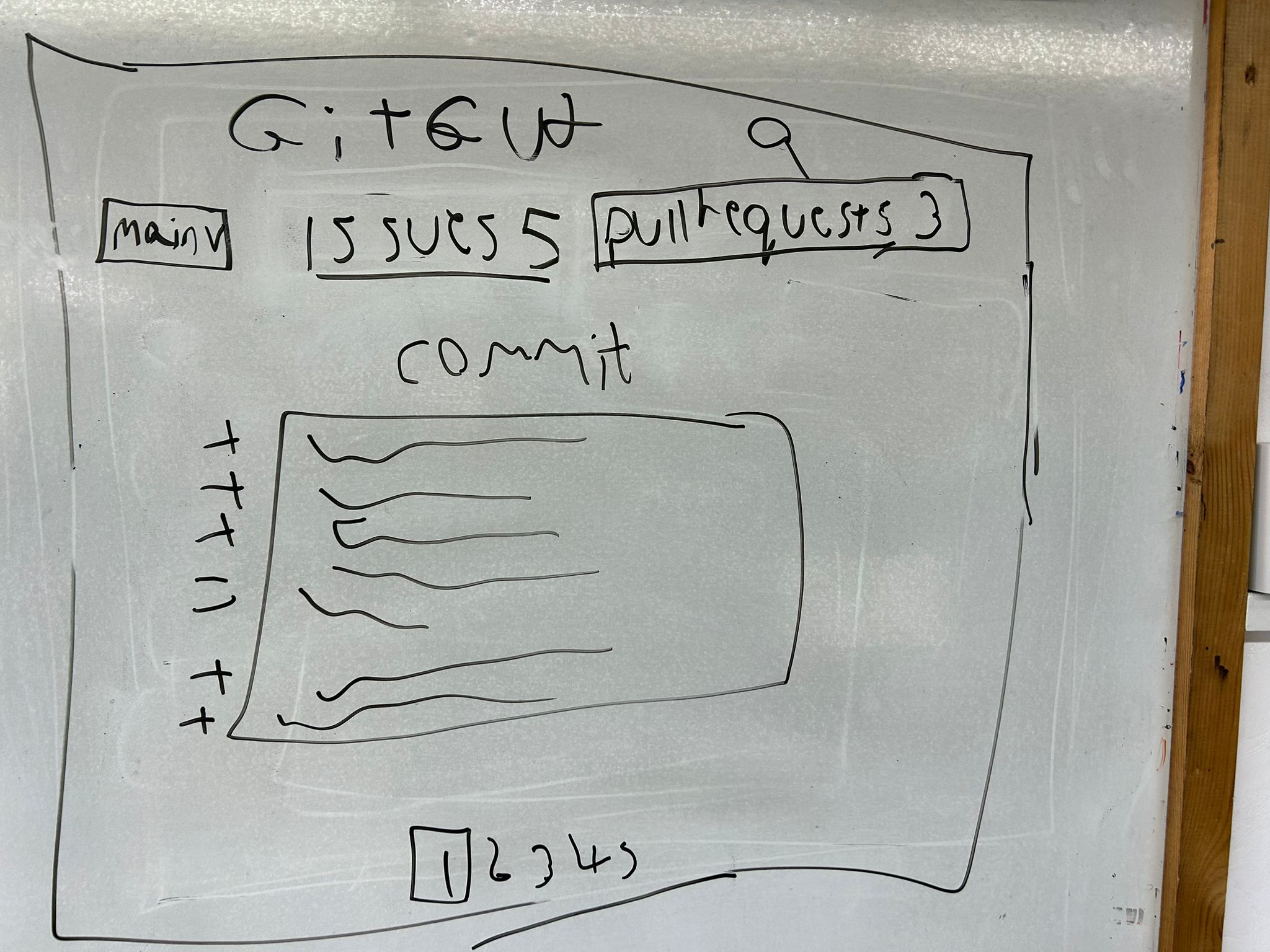
**מסך צפייה ב commits**

כל ה commits messages מוצגים ברשימה



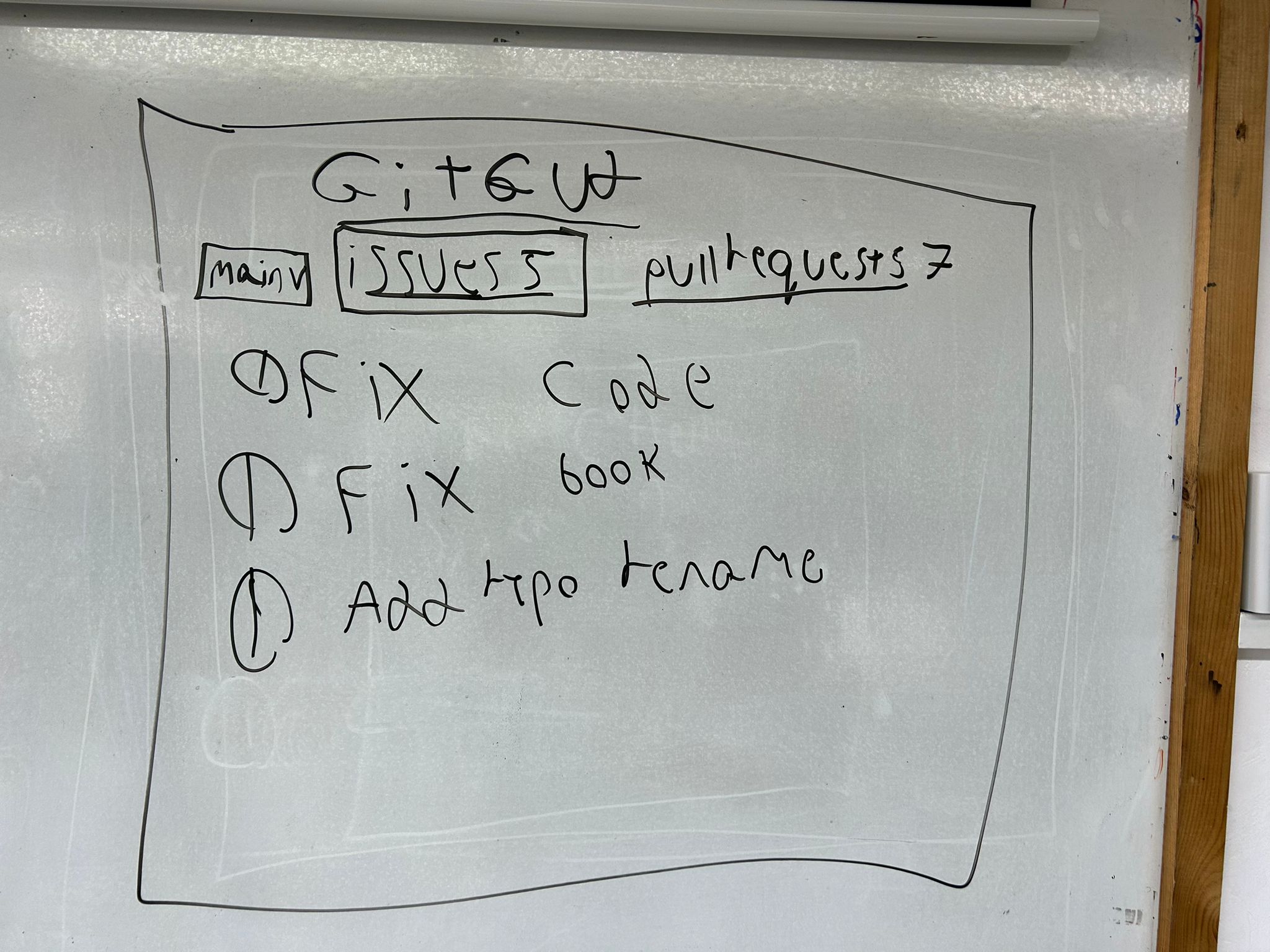
**מסך צפייה ב commit**

מציג את ה commit message ואת השינויים



**מסך צפייה בissues**

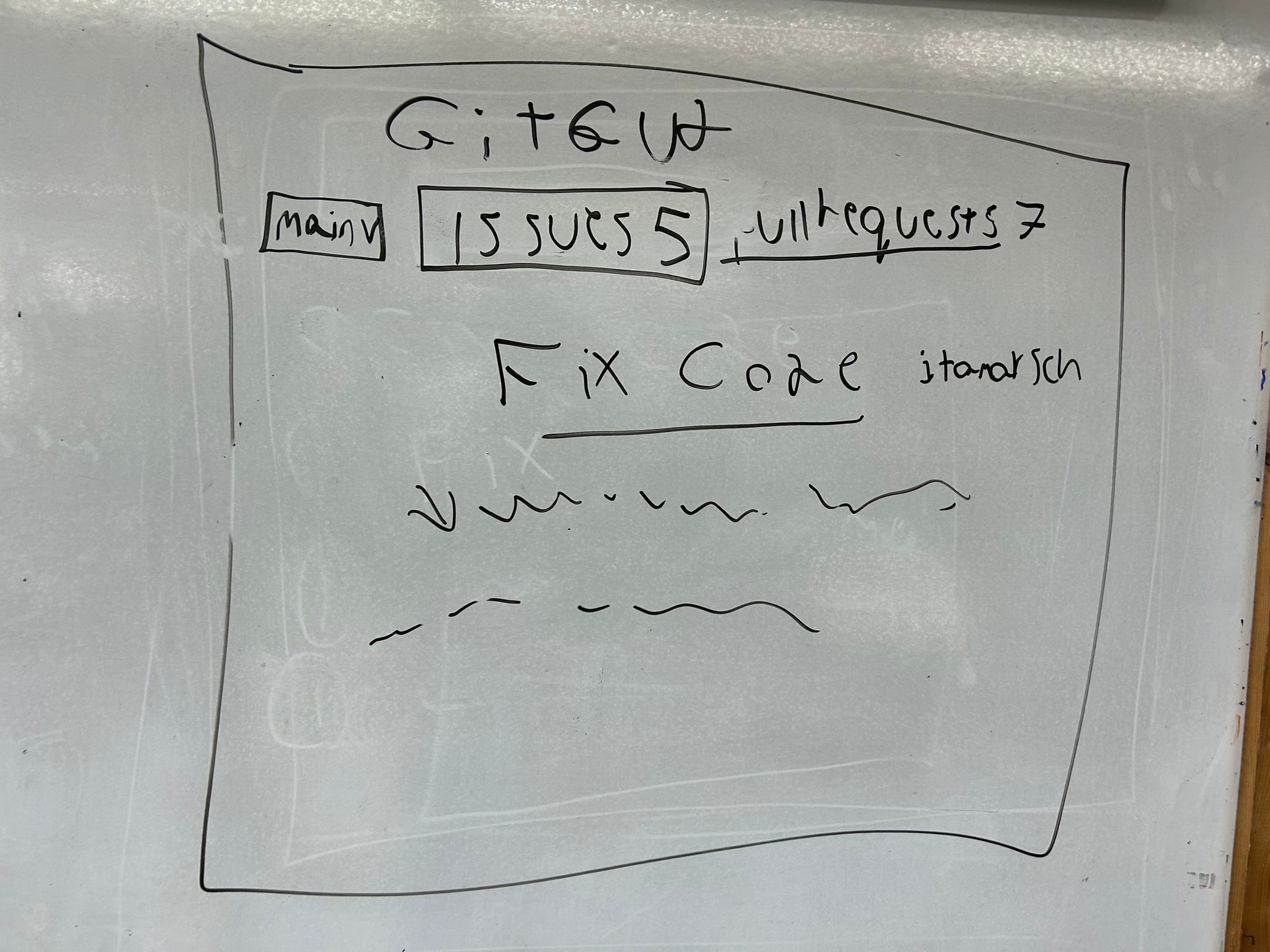
מאפשר לחיצה על issue בשביל צפייה בתוכן



**מסך צפייה בissues**

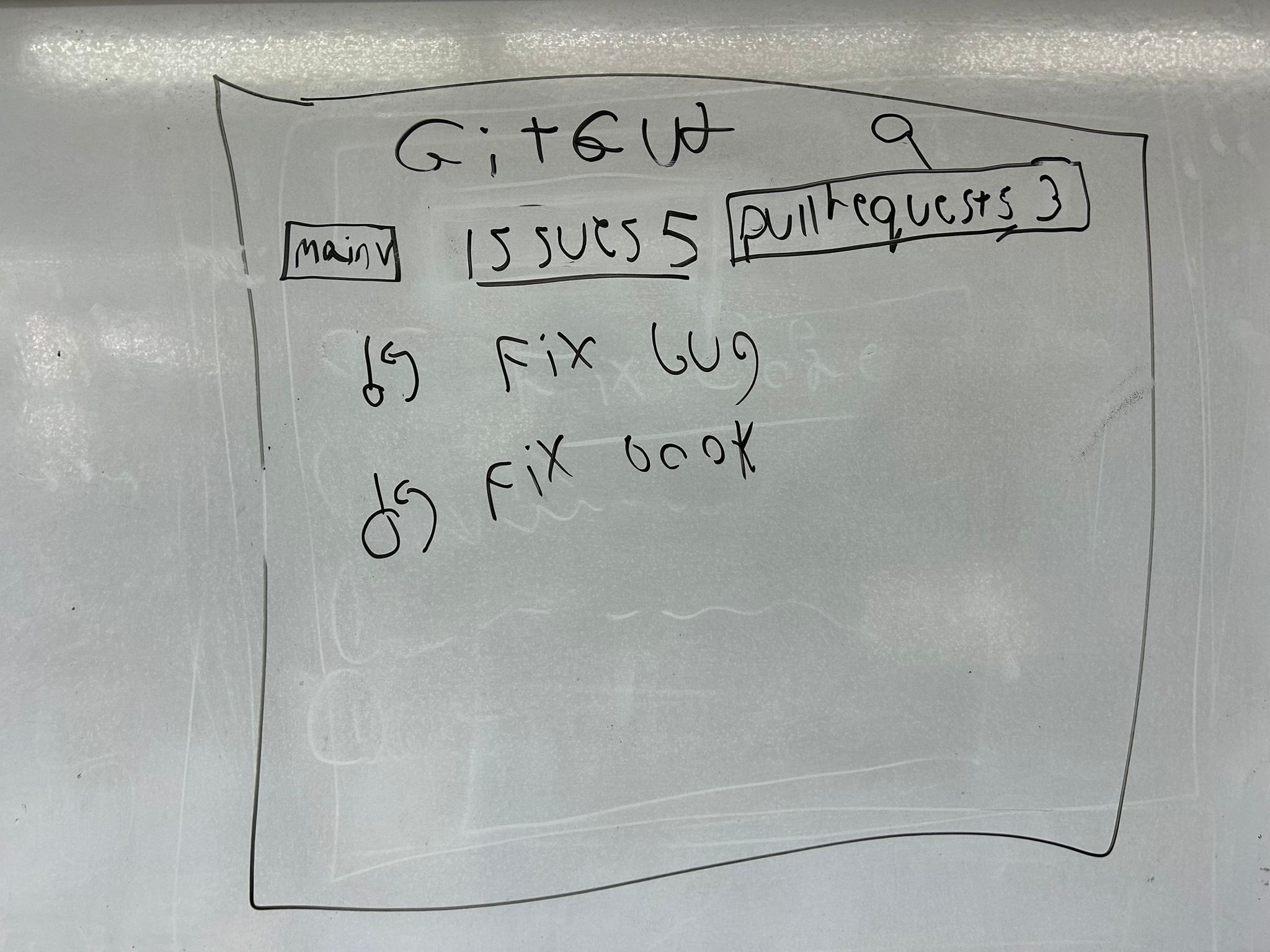
מציג את התוכן של issue

מאפשר עריכה ומחיקה של ה issue



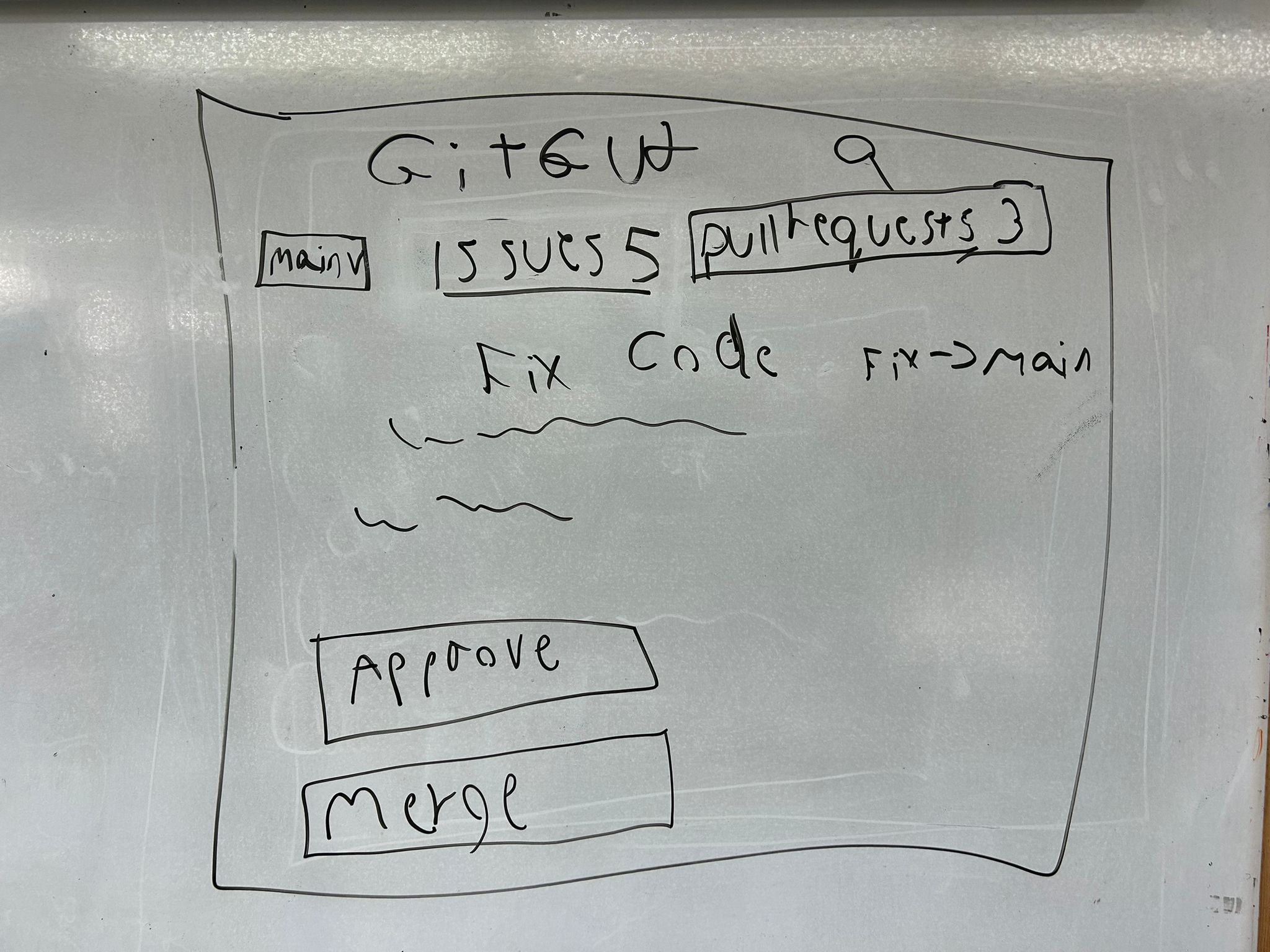
**מסך צפייה ב pull requests**

מציג את הpull requests הנוכחיים



**מסך pull request**

מאפשר merge, approve, צפייה בשינויים ומחיקה ועריכה של ה pull request



פרוטוקול תקשורתL

* הודעות תמיד עוברת דחיסה ואז הצפנה בשליחה

הודעות בסוקט קבצים:

* תמיד עובר אורך ההודעה ב 8 בתים
* בקשות יהיו בjson ותשובות יהיו קבצים

| משמעות | שם ההודעה | ממי למי | שדות | דוגמא |
| --- | --- | --- | --- | --- |
| בקשת הרשמה | register | לקוח לשרת | String username  String password (hashed)  String sshKey | {  “type”: “register”,  “username”: “itamarsch”,  “password”: “c8cadbfbdff908ae9e0ce34bfb735cc62061faa31f94583a8e26c51f2d61b637”,  “sshKey”: “ssh-ed25519 dsfdsfndsfkfsdkfdsSSAWWDDCCSCScndkscslcsls schwartzitamar7@gmail.com”  } |
| תשובת הרשמה | registerS | שרת ללקוח | String connectionToken | {  “type”: “register”,  “error”: “Username already exists”,  }  {  “type”: “register”,  “connectionToken”: “sfdfdsddsvvsddssd”  } |
| התחברות: | login | לקוח לשרת | String username  String password (hashed) | {  “type”: “login”,  “username”: “itamarsch”,  “password”: “c8cadbfbdff908ae9e0ce34bfb735cc62061faa31f94583a8e26c51f2d61b637”,  } |
| תשובת התחברות | loginS | שרת ללקוח | String connectionToken | {  “type”: “loginS”,  “error”: “Incorrect password”,  }  {  “type”: “loginS”,  “connectionToken”: “sfdfdsddsvvsddssd”  } |
| בדיקת התחברות | validateConnection | לקוח לשרת | String connectionToken | {  “type”: “checkToken”,  “connectionToken”: “ sfdfdsddsvvsddssd”  } |
| תשובה לבדיקת התחברות | validateConnectionS | שרת ללקוח |  | {  “error”: “no such token”,  }  {  } |
| יצירת פרויקט | createRepo | לקוח לשרת | String name,  String visibility (“public” | “private”) | {  “type”: “createRepo”,  “repoName”: “GitGud”,  “visibility”: “public”,  ”connectionToken”: “sfdfdsddsvvsddssd”  } |
| תשובת יצירת פרויקט | createRepoS | שרת ללקוח |  | {  “type”: “createRepoS”,  “error”: “Repository already exists”  },  {  “type”: “createRepoS”,  }, |
| רשימת branches | branches | לקוח לשרת | String repo (“user/reponame”) | {  “type”: “branches”,  “repo”: “ itamarsch/GitGud”  } |
| תשובת רשימת branches | branchesS | שרת ללקוח | [String] branches | {  “type”: “branchesS”,  “branches”: [“main”, “feature-1”]  } |
| צפייה בקובץ בפרויקט | viewFile | לקוח לשרת | String repo (“user/reponame”)  String filePath  String branch | {  “type”: “viewFile”,  “repo”: “itamarsch/GitGud”  “filePath”: “assets/video.mp4”  “connectionToken”:”sfdfdsddsvvsddssd”,  “branch”:”main”  } |
| תשובה לצפייה בקובץ בפרויקט | viewFileS | שרת ללקוח | int port  String passwordForFileRequests | {  “type”: “viewFileS,  “port”:55123,  “passwordForFileRequests”:”dkfldgmfgdfmw”  }  {  “type”: “viewFileS”  “error”: “Invalid filepath”  } |
| בקשת קובץ | getFile | לקוח לסוקט קבצים | String fileSocketToken | {  “type”: “getFIle”,  “fileSocketToken”: dkfldgmfgdfmw”  } |
| תשובת קובץ | getFileS | סוקט קבצים ללקוח |  | print(“Hello world!”) |
| צפייה ברשימת קבצים בתקיה בפרויקט | projectDirectory | לקוח לשרת | String directory  String branch  String repo (“user/reponame”) | {  “type”:”projectDirectory”,  “directory”: “lib/”  “repo”: “itamarsch/GitGud”,  “branch”: “main”,  “connectionToken”: “sfdfdsddsvvsddssd”  } |
| תשובה לצפייה ברשימת קבצים בתקייה בפרויקט | projectDirectoryS | שרת ללקוח | [String] files | {  “type” : “projectDirectoryS”  “error”: “Directory does exits”’  }  {  “type” : “projectDirectoryS”,  “files”: [“main.py”, “log.py”, “assets/”]  } |
| צפייה בכל הקומיטים שנעשו בפרוייקט | commits | לקוח לשרת | String repo (“user/reponame”)  String branch  int page //Commits will be shown in chunks (pages) | {  “type”: “commits”,  “repo”: “itamarsch/GitGud”,  “page”: 1,  “branch”: “main”,  “connectionToken”:”sfdfdsddsvvsddssd”  } |
| תשובה של כל ה commits | commitsS | שרת ללקוח | [Commit] commits  Commit:  String commitMessage  String commitHash  String commiterName  String date | {  “type”: “commitsS”  “error”: “Insufficient permissions”  }  }  {  “commits”: [  {“commitMessage”: “Ready. Set. Go!”,  “commitHash”: “ddfddafgg”,  “commiterName”:“itamar schwartz”,  “date”: ”7-05-24”},  {“commitMessage”: “Fix bug”,  “commitHash”: “fgdfzfds”,  “commiterName”:“itamar schwartz”,  “date”: ”8-05-24”}  ] |
| צפייה בשינויים שנעשו ב commit | diff | לקוח לשרת | String commitHash  String repo (“user/reponame”) | {  “type”:”diff”,  “commitHash”: “fgnfdkdnsff”  “repo”: “itamarsch/GitGud”,  “connectionToken”: “sfdfdsddsvvsddssd”  } |
| קבלת שינויים שנעשו ב commit | diffS | שרת ללקוח | int port  String passwordForFileRequests | {  “type”: “diffS”,  “port”:55123,  “passwordForFileRequests”:”dkfldgmfgdfmw”  } |
| יצירת Issue | createIssue | לקוח לשרת | String repo (“user/repo”)  String issueName  String issueContent | {  “type”: “createIssue”,  “repo”: “itamarsch/GitGud”,  “issueName”:”Project not finished”,  “issueContent”: “Project needs to be finished by end of the year”  “connectionToken”: “sfdfdsddsvvsddssd”  } |
| תשובה ליצירת issur | createIssueS | שרת ללקוח |  | {  “type”: “createIssueS”  }  {  “type”: “createIssueS”,  “error”: “Insufficient permissions”  } |
| צפייה ב issues | viewIssues | לקוח לשרת | String repo (“user/repo”) | {  “type”: “viewIssues”,  “repo”: “itamarsch/GitGud”,  “connectionToken”: “sfdfdsddsvvsddssd”  } |
| תשובה לצפייה בissues | viewIssuesS | שרת ללקוח | [Issue] issues  Issue:  String issueCreator  String issueName  String issueContent  int issueID | {  “type”: ”viewIssuesS”  “error”: “Repository doesn’t exist”  }  {  “type”: “viewIssuesS”  “issues”: [  {  “issueCreator”:”Itamar Schwartz”,  “issueName”:  “fds”,  “issueContent”:”ddssgadgdfsg”  “issueID”:  200  },  {  “issueCreator”:”Itamar Schwartz”,  “issueName”:  “fds”,  “issueContent”:”ddssgadgdfsg”  “issueID”:  100  }  ]  } |
| מחיקה של issue | deleteIssue | לקוח לשרת | Int issueID | {  “type”: “deleteIssue”,  “issueID”: 100,  “connectionToken”: “sfdfdsddsvvsddssd”  } |
| תשובה למחיקה issue | deleteIssueS | שרת ללקוח |  | {  “type”: “deleteIssueS”  “error”:”Issue doesn’t exist”  }  {  “type”: “deleteIssueS”  } |
| עריכת Issue | updateIssue | לקוח לשרת | int issueID  String newIssueName  String newIssueContent  int id | {  “type”: “updateIssue”,  “issueID”: 100,  “newIssueName”: “Ifgdgdgfdg,  “newIssueContent”:  “fgdhgdhgg”,  “connectionToken”: “sfdfdsddsvvsddssd”,  “id”: 20  }  } |
| תשובת עריכת issue | updateIssueS | שרת ללקח |  | {  “type”: “updateIssueS”  }  {  “type”: “updateIssueS”,  “error”: “Issue doesn’t exists”  } |
| חיפוש repositories | searchRepositories | לקוח לשרת | String searchQuery | {  “type”: “searchRepositories”,  “searchQuery“: “Git”,  “connectionToken”: “sfdfdsddsvvsddssd”  } |
| תשובה לחיפוש repositories | searchRepositoriesS | שרת ללקוח | [String] repositories | {  “repositories”: [  “itamarsch/GitGud”,  “micorosoft/Github”,  “linusTorvalds/Git”  ]  } |
| יצירת pull request | createPullRequest | לקוח לשרת | String repo (“user/repo”)  String pullRequestName  String fromBranch  String intoBranch | {  “type”: “createPullRequest”,  “repo”: “itamarsch/GitGud”,  “pullRequestName”:”Implement issues”,  “fromBranch”: “issuesFeature”,  “intoBranch”: “main”  “connectionToken”: “sfdfdsddsvvsddssd”  } |
| תשובה ליצירת pull request | createPullRequestS | שרת ללקוח |  | {  “type”: “createPullRequestS”  }  {  “type”: “createPullRequestS”,  “error”: “Branches don’t exist”  } |
| צפייה ב pull request | viewPullRequests | לקוח לשרת | String repo (“user/repo”) | {  “type”: “viewPullRequests”,  “repo”: “itamarsch/GitGud”,  “connectionToken”: “sfdfdsddsvvsddssd”  } |
| תשובה לצפייה ב pull requests | viewPullRequestsS | שרת ללקוח | [PullRequest] pullRequests  PullRequest:  String pullRequestCreator  String pullRequestName  String fromBranch  String intoBranch  int pullRequestID  boolean approved | {  “type”: ”viewPullRequestsS”  “error”: “Repository doesn’t exist”  }  {  “type”: “viewPullRequestsS”  “pullRequests”: [  {  “pullRequestCreator”:”Itamar Schwartz”,  “pullRequestName”:  “fds”,  “fromBranch”:”issueFeature”,  “intoBranch”: “master”,  “pullRequestID”:  200,  “approved”: false  },  ]  } |
| מחיקה של pull request | deletePullRequest | לקוח לשרת | Int pullRequestID | {  “type”: “deletePullRequest”,  “pullRequestID”: 100,  “connectionToken”: “sfdfdsddsvvsddssd”  } |
| תשובה למחיקה pull requst | deletePullRequestS | שרת ללקוח |  | {  “type”: “deletePullRequestS”  “error”:”Pull request doesn’t exist”  }  {  “type”: “deletePullRequestS”  } |
| עריכת Pull request | updatePullRequest | לקוח לשרת | int pullRequestID  String? newPullRequestName  String? newFromBranch  String? newIntoBranch | {  “type”: “updateIssue”,  “pullRequestID”: 100,  “newPullRequestName”: “Ifgdgdgfdg,  “connectionToken”: “sfdfdsddsvvsddssd”  } |
| תשובת עריכת Pull request | updatePullRequest | שרת ללקח |  | {  “type”: “updatePullRequest”  }  {  “type”: “updatePullRequest”,  “error”: “PullRequest doesn’t exists”  } |
| בקשת שינויים של pull request | pullRequestDiff | לקוח לשרת | Int pullRequestID | {  “type”: “pullRequestDiff”,  “pullRequestID”: 4,  “connectionToken”: “fkdfsdlsg”  } |
| תשובת בקשת שינויים לpullrequest | pullRequestDiffS | שרת ללקוח | int port  String passwordForFileRequests | {  “type”: “pullRequestDiffS”,  “port”:55123,  “passwordForFileRequests”:”dkfldgmfgdfmw”  } |
| כל הcommits שנעשו בpr | prCommits | לקוח לשרת | Int id  int page | {  "type": "prCommits",  "id": 5,  "page": 10,  } |

**בסיס נתונים:**

אני אשתמש בdb של postgres בשביל לשמור את המידע על השרת.

User:

username: str (primary key)

password: str (hash)

id: int (primary key)

Unique keys: (id, username)

Repository:

id: int (primary key)

public: bool

user\_id: int

name: str

Unique keys: (id), (user\_id,name)

Foreign keys: (user\_id → User.id)

Issue:

id: int (primary key)

user\_id: int

repo\_id: int

title: str

content: str

Unique keys: (id)

Foreign keys: (user\_id → User.id), (repo\_id → Repository.id)

Pull request:

id: int (primary key)

user\_id: int

repo\_id: int

title: str

from\_branch: str

into\_branch: str

Unique keys: (id)

Foreign keys: (user\_id → User.id), (repo\_id -> Repository.id)

**קבצים:**

צד שרת:

* כל קבצי הפרויקטים של הלקוחות בתוך gitolite
* קבצי ההרשאות של מנהל ה Git

צד לקוח:

* שמור קובץ של מפתח התחברות -

%USERPROFILE%\git\_gud.txt

מודולים שאני מייבא:

|  |  |
| --- | --- |
| שם | לטובת |
| typing/typing\_extensions | שימוש נכון ב types לחתימות הפונקציות בפרויקט |
| Fernet (cryptography) | כל מה ששקשור להצפנה |
| socket | כל מה שקשור לתקשורת ברשת |
| os | התעסקות עם קבצים |
| sys | התעסקות עם המערכת הפעלה |
| git | תקשורת עם פרויקטי הלקוח |
| psycopg2 | תקשורת עם postgres |
| wxpython | ממשק המשתמש |
| compross | דחיסה |
| pygments | syntax highlighting בלקוח |
| pyperclip | להעתיק בלקוח |
| python-dotenv | שמירת קבצי קונפיגורציה |
| fuzzywuzzy | חיפוש של מילים דומות |

Server

ServerProtocol

| טענת יציאה | טענת כניסה | שם הפעולה |
| --- | --- | --- |
| Dictionar­y (JSON) of requests | String json – Message to unpack | unpack |
| JSON that fits protocol requirements | String error – Error message  String type - messageType | packErrorS |
| JSON that fits protocol requirements | String connectionToken – Token for future connections | packRegisterS |
| JSON that fits protocol requirements | String connectionToken - Token for future connections | packLoginS |
| JSON that fits protocol requirements |  | packCheckTokenS |
| JSON that fits protocol requirements |  | packCreateRepoS |
| JSON that fits protocol requirements | [String] branches – Branches of repository | packBranchesS |
| JSON that fits protocol requirements | int port – Port on file socket to connect to  String passwordForFileRequests – Password for file socket request | packViewFileS |
| JSON that fits protocol requirements | String fileContent – file contents to send in message | packGetFileS |
| JSON that fits protocol requirements | [String] files – File names in directory | packProjectDirectoryS |
| JSON that fits protocol requirements | [Commit] commits – Commits of repository | packCommitsS |
| JSON that fits protocol requirements | int port – Port on file socket to connect to  String passwordForFileRequests – Password for file socket request | packDiffS |
| JSON that fits protocol requirements |  | packCreateIssueS |
| JSON that fits protocol requirements | [Issue] issues – Issues from repo | packViewIssuesS |
| JSON that fits protocol requirements |  | packDeleteIssueS |
| JSON that fits protocol requirements |  | packUpdateIssueS |
| JSON that fits protocol requirements | [String] repositories – Repositories that fir search query | packSearchRepositoriesS |
| JSON that fits protocol requirements |  | packCreatePullRequestS |
| JSON that fits protocol requirements | [PullRequest] pullRequests – Pull requests from repo | packViewPullRequestsS |
| JSON that fits protocol requirements |  | packDeletePullRequestS |
| JSON that fits protocol requirements |  | packUpdatePullRequestS |
| JSON that fits protocol requirements | List[str] Repos | pack\_search\_repo |

GitManager

המחלקה אחראית על תקשורת עם מנהל הgit, תקשורת נעשת בתוך repository בשם gitolite-admin כשעושים אליו push הקונפיגורציה מתעדכנת

|  |  |  |  |
| --- | --- | --- | --- |
| משמעות | | שם השדה | |
| התקיה שבה הפרויקט נמצא | | String path | |
| אובייקט של GitPython שמייצג repo | | Repo repo | |
| טענת יציאה | טענת כניסה | | פעולה |
| Exception – if project doesn’t exist  Creates connection to repo | String path | | \_\_init\_\_ |
| Adds an SSH key to gitolite for a specific user. | self  String user\_name,  String ssh\_key | | add\_ssh\_key |
| Commits changes in manager repository | self  String commit\_message  [String] files\_to\_commit | | \_commit\_changes |
| Create a new repository in gitolite with the specified name for a given user. | self  String repo\_name  String user  bool public | | create\_repo |

DB

המחלקה אחראית על תקשורת עם ה database

|  |  |
| --- | --- |
| משמעות | שם בשדה |
| אובייקט של Psycopg2 בשביל תקשורת עם postgres | Connection conn |

|  |  |  |
| --- | --- | --- |
| טענת יציאה | טענת כניסה | שם הפעולה |
| Exception – user already exists  Returns: int: The id of the newly added user. | self,  String username  String passwordHash | addUser |
| Repository | self, String repo (“user/repo”) | repoByName |
| Optional[Tuple[int, str, bool]]: A tuple containing the repository's id, name, and public status if found, or None if the repository is not found. | self, int user\_id, String reponame, bool public | createRepo |
| Optional[int]: The ID corresponding to the username if found, otherwise None. | String username | username\_to\_id |
| [Issue]: Retrieve a list of issues for the given repository ID. | self, int repoID | issues |
| Creates an issue in the database with the provided user ID, repository ID, title, and content. | self, int repoID, String content, int user\_id, String title | createIssue |
| Deletes an issue from the database based on the given issue\_id. | self, int issueID | deleteIssue |
| Update title, and content for the issue with the given issue\_id, . | self, int issueID, String content, String newTitle | updateIssue |
| Creates a pull request with the specified title, from\_branch, into\_branch, repo\_id, and user\_id. | self, String title, String from\_branch, String into\_branch, int repo\_id,int user\_id | createPullRequest |
| Delete a pull request from the database. | self, int pullRequestID | deletePullRequest |
| Updates a Pull Request record in the database with the provided title, from\_branch, and into\_branch for a given id. | self, int pullRequestID,  String title,  String from\_branch,  String into\_branch,  int repo\_id, | updatePullRequest |
| [PullRequest] pullrequests  Retrieves pull requests for a given repo\_id. | self, int repo\_id | pullRequests |
| bool: Check if a user exists in the database. | self, String username | user\_exists |
| [Repositories] Retrieve all public repositories from the database, including the repository id, owner's username, and repository name. | self, | all\_repos |
| bool correct | self, String username, String passwordHash | validateUser |
| Optional[Tuple[str,str]]: A function to retrieve the repository and repository owner of a given issue. | self, int issue\_id | repo\_and\_repo\_owner\_of\_issue |
| Optional[Tuple[str, str]]: Retrieves the repository name and owner username associated with the given pull request ID. | Self, int pr\_id | repo\_and\_repo\_owner\_of\_pr |

git-utils

מודול של פונקציות עזרה עם git

|  |  |  |
| --- | --- | --- |
| תענת יציאה | תענת כניסה | שם הפעולה |
| [Commit]: Returns a list of commits between two branches of a given repository. | Repo repo  String from\_branch  String into\_branch  int page | commits\_between\_branches |
| Json: Creates a json object from the output of git diff | String git\_diff\_output | get\_diff\_json |

Encryption (Server)

המחלקה אחראית על כל מה שקשור להצפנה בצד השרת

|  |  |
| --- | --- |
| משמעות | שם המשתנה |
| Secret key generated only for server | secret\_key: int |
| Encryption key combined from client, server and public keys | encryption\_key: int | null |
| Fernet object used for encrypting | fernet: Fernet | null |

|  |  |  |
| --- | --- | --- |
| טענת יציאה | טענת כניסה | שם הפעולה |
| Calculates encryption key based on B key | self, client\_mixed\_key: int | set\_encryption\_key |
| String message  Returns message of keys for client | self | get\_initial\_public\_message |
| bool finished encryption | self | finished\_encryption |
| Uses p g and secret a to calculate A | self | \_calculate\_mixed\_key |

ServerComm

המחלקה אחראית על תקשורת בצד השרת

|  |  |
| --- | --- |
| משמעות | שם המשתנה |
| The socket of the server | server\_socket: socket.socket |
| Current open sockets | open\_socekts: [address, (socket, EncryptionState)] |
| Logic message queue | logic\_queue: Queue |
| Whether the server is running | running: bool |

|  |  |  |
| --- | --- | --- |
| טענת יציאה | טענת כניסה | שם הפעולה |
| Sends message to socket and closes it | self, ip: str, message: str | send\_and\_close |
| Bytes compress bytes into bytes | data: bytes | compress\_bytes |
| Bytes: Compress str into bytes | data: str | compress\_str |
| String: Decompresses bytes into a str | data: bytes | decompress\_bytes |
| Receives length size amount of bytes to receive the length and receives the length amount of bytes for the data | Soc: socket, length\_size: int | recv |
| Sends bytes with length at the start with size of length size | soc: socket, length\_size: int | send |
| Optional[Address]: Returns the address of the socket if it is part of the open\_sockets | self, soc: socket | \_get\_addr\_of\_socket |
| Start the server on the port | self, port: int | \_listen |
| Disconnects client based on address | self, address: Address | \_disconnect\_client |
| Read sockets and handle new client connections or received messages. | self, rlist: List[socket.socket] | \_read\_sockets |
| Handle receiving and processing messages from a socket connection. | self, soc: socket.socket, addr: Address | \_on\_message\_receive |
| A function to handle receiving encryption responses from a socket. | self, soc: socket.socket, addr: Address | \_on\_receive\_encryption |
| Initializes a new client connection with the given socket and address. | self, soc: socket.socket, addr: Address | \_new\_client |
| Starts thread for server | self, port: int | start\_listeneing |

FileComm

המחלקה אחראית על התקשורת של קבצים ברשת

|  |  |
| --- | --- |
| משמעות | שם המשתנה |
| Token for validation of connection to file comm | token: str |
| Content of file to send | data: bytes |

|  |  |  |
| --- | --- | --- |
| טענת יציאה | טענת כניסה | שם הפעולה |
| Starts thread for handling file sending | self | start\_listening |
| Returns port of file comm that was selected randomly by os | self | get\_port |
| Listen for incoming connections, establish encryption, receive file request, validate file request, send file, then close the connection. | self | \_listen |

SSHKeyValidation

|  |  |  |
| --- | --- | --- |
| טענת יציאה | טענת כניסה | שם הפעולה |
| Validate a public ssh key string.  Bool: true if the key is valid | pub\_key: str | validate\_pubkey |

ServerLogic

|  |  |
| --- | --- |
| משמעות | שם המשתנה |
| DB communication object | db: DB |
| Object for | git\_manager: GitManager |
| Object for network communication on the server | server\_com: ServerComm |
| Queue of json messages from clients | queue: asyncio.Queue |
| Dictionary of all authentication tokens for clients | authentication\_tokens: Dict[token, username] |

|  |  |  |
| --- | --- | --- |
| טענת יציאה | טענת כניסה | שם הפעולה |
| Process a request from the queue, apply an action, and send the response back to the client. | self | tick |
| A function to apply a given action to a JSON input according to the type assuming the keys are validated and return the result as JSON. | self, json: Json | apply\_action |
| Returns all actions as a dictionary from the type to the function and the required keys | self | get\_actions |
| String: Generate a new connection token for a given username. | self, username: str | generate\_new\_connection\_token |
| Validate the repository request by checking if the user and repo are valid.  Returns: Union[Json, Tuple[int, str, bool]]: Either repository data if valid, or an error message | self, repo: str, connectionToken: str | validate\_repo\_request |
| Create a new client in the application and sends token | self, json: Json | register |
| Sends new token to client | self, json: Json | login |
| Send token validity to client | self, json: Json | validate\_token |
| Creates repository in file system | self, json: Json | createRepo |
| Sends branches of repository | self, json: Json | branches |
| Sends port and token to client and Starts task for file receiving | self, json: Json | viewFile |
| Sends file names in directory in project | self, json: Json | projectDirectory |
| Generate a JSON representation of a list of Git commits. | commits: List[GitCommit] | pack\_git\_commits |
| Sends commit in repository in branch | self, json: Json | commits |
| A function that processes and returns commits related to a pull request. | self, json: Json | pr\_commits |
| Sends port and token to client and Starts task for diff receiving | self, json: Json | diff |
| Creates issue | self, json: Json | createIssue |
| Sends issues in project | self, json: Json | viewIssues |
| Deletes issues | self, json: Json | deleteIssue |
| Updates issue | self, json: Json | updateIssue |
| Validate the given issue or PR with the provided ID, connection token, and issue or PR enum. | self, id: int, connection\_token: str, issue: IssuePr | validate\_issue\_or\_pr |
| Searches for public repositories that fir searcg query | json: Json | searchRepositories |
| Creates pull request | self, json: Json | createPullRequest |
| Returns pull requests in project | self, json: Json | viewPullRequests |
| Deletes pull request | self, json: Json | deletePullRequest |
| Updates Pull Requst | self, json: Json | updatePullRequest |
| Validate the branches of a pull request. and deletes if invalid | self, repo: Repo, id: int, into\_branch: str, from\_branch: str | validate\_pr\_branches |
| A function to calculate the difference between two branches in a pull request. | self, json: Json | pr\_diff |
| Generate a list of branch names for the given repository. | repo: Repo | branches\_of\_repo |

RepoClone

|  |  |  |
| --- | --- | --- |
| טענת יציאה | טענת כניסה | שם הפעולה |
| Repo: Clones repo into cache folder if it doesn't exist and pulls branch if it does | repo\_name: str, branch: str = "" | repo\_clone |

Client

Token file

|  |  |  |
| --- | --- | --- |
| טענת יציאה | טענת כניסה | שם הפעולה |
| Saves the token to a file. | token: str | save\_token\_file |
| Reads the token file and returns its contents as a string, or returns None if the file does not exist. |  | read\_token\_file |

MainFrame

|  |  |
| --- | --- |
| משמעות | שם המשתנה |
| Stack of screens | Screens: List[BaseScreen] |
| Class for communication with the server | client\_com: ClientCom |
| Sizer of screen | Sizer: wx.BoxSizer |

|  |  |  |
| --- | --- | --- |
| טענת יציאה | טענת כניסה | שם הפעולה |
| Pushes a new screen onto the screen stack and sets it as the active screen. Hides the previous screen if there is one. | Self, screen: BaseScreen | push\_screen |
| Pops the top screen, hides it, and then shows the new top screen. | self | pop\_screen |
| Changes the current screen to the one provided by the panel parameter. | self | change\_screen |

BaseScreen

Base class for all screens

|  |  |
| --- | --- |
| משמעות | שם המשתנה |
| MainFrame parent of all screen for pushing and poping | Parent: Mainframe |

|  |  |  |
| --- | --- | --- |
| טענת יציאה | טענת כניסה | שם הפעולה |
| Function that can be overrided by the child class. This function is called once a screen is reloaded after a pop of a screen on top of it | self | on\_reload |
| A method for getting the parent of the BaseScreen which os always a MainFrame | self | get\_parent |
| Function that should be overrided by the child class. Adds all ui widgets to the main sizer | self | add\_children |

HashPassword

|  |  |  |
| --- | --- | --- |
| טענת יציאה | טענת כניסה | שם הפעולה |
| Compute the SHA-256 hash of the input password and return the base64 encoded hash value as a string. | password | hash |

GuiRunReqeust

|  |  |  |
| --- | --- | --- |
| טענת יציאה | טענת כניסה | שם הפעולה |
| A function that sends a file request to the GUI, runs the request, and calls the on\_finished callback with the file content. | panel: BaseScreen, request: Json, on\_finished: Callable[[bytes], None] | gui\_request\_file |
| A function to run a request in a GUI environment and handle the response asynchronously.  Shows an error box if there is an error | panel: BaseScreen, request: Json, on\_finished: Callable[[Json], None], message\_box\_error=True, | gui\_run\_request |

ClientProtocol

| שם הפעולה | טענת כניסה | טענת יציאה |
| --- | --- | --- |
| packRegister | String username – New username  String password (hashed) – Hash of user password  String sshKey – SSH Key for | JSON that fits protocol requirements |
| packLogin | String username  String password (hashed) | JSON that fits protocol requirements |
| packCheckToken | String connectionToken – token for validity checking | JSON that fits protocol requirements |
| packCreateRepo | String name,  String visibility (“public” | “private”),  String connectionToken | JSON that fits protocol requirements |
| packBranches | String repo (“user/reponame”),  String connectionToken | JSON that fits protocol requirements |
| packViewFile | String repo (“user/reponame”)  String filePath,  String connectionToken | JSON that fits protocol requirements |
| packGetFile | String fileSocketToken | JSON that fits protocol requirements |
| packProjectDirectory | String directory  String branch  String repo (“user/reponame”),  String connectionToken | JSON that fits protocol requirements |
| packCommits | String repo (“user/reponame”)  String branch  int page //Commits will be shown in pages,  String connectionToken | JSON that fits protocol requirements |
| packDiff | String commitHash  String repo (“user/reponame”),  String connectionToken | JSON that fits protocol requirements |
| packCreateIssue | String repo (“user/repo”)  String issueName  String issueContent,  String connectionToken | JSON that fits protocol requirements |
| packViewIssues | String repo (“user/repo”),  String connectionToken | JSON that fits protocol requirements |
| packDeleteIssue | Int issueID,  String connectionToken | JSON that fits protocol requirements |
| packUpdateIssue | int issueID  String | null newIssueName  String | null newIssueContent,  String | null connectionToken | JSON that fits protocol requirements |
| packSearchRepositories | String searchQuery,  String connectionToken | JSON that fits protocol requirements |
| packCreatePullRequest | String repo (“user/repo”)  String pullRequestName  String fromBranch  String intoBranch,  String connectionToken | JSON that fits protocol requirements |
| packViewPullRequests | String repo (“user/repo”),  String connectionToken | JSON that fits protocol requirements |
| packDeletePullRequest | Int pullRequestID,  String connectionToken | JSON that fits protocol requirements |
| packUpdatePullRequest | int pullRequestID  String | null newPullRequestName  String | null newFromBranch  String | null newIntoBranch,  String | null connectionToken | JSON that fits protocol requirements |
| packPullRequestCommits | Int pullRequestID,  String connectionToken  int page | JSON that fits protocol requirements |
| packPullRequestDiff | Int pullRequestID | JSON that fits protocol requirements |

ClientComm

המחלקה אחראית על תקשורת עם השרת בצד הלקוח

|  |  |  |
| --- | --- | --- |
| טענת יציאה | טענת כניסה | שם הפעולה |
| EncryptionState: A function to exchange keys with a socket using encryption and return the encryption state. | self, soc: socket.socket | \_exchange\_keys |
| Json: Runs a request with the provided data using socket communication. | self, data: Json | run\_request |
| Bytes: Runs a file request using the provided token and port number. | self, token: str, port: int | file\_request |

מסכים

1. login: מסך התחברות למשתמש
2. register: מסך הרשמה
3. ssh\_help: מסך שאפשר להגיע אליו ממסך הregister וקיים בו הסבר על איך ליצור מפתח ssh משלך
4. repo\_screen: מסך "ראשי" שבו אפשר לחפש repository, לצפות קבצים, לעבור למסכים של pull requests וissues ליצור repository
5. pull\_requests: מסך שבו אפשר לצפות בכל הpull requests של repository
6. pr\_editor: מסך שאחראי על עריכת pull request זאת אומרת גם בשביל ליצור וגם בשביל לערוך
7. issues: מסך שאחראי על להציג את כל הissues על פרויקט
8. issue\_editor: מסך מקביל ל pr\_editor בשביל issues
9. issue: מסך בשביל צפייה בתוכן של issue
10. file\_screen: מסך שאחראי על הצגת התוכן של קובץ, וצביעת הטקסט על פי שפת התכנות
11. commits: מסך שאחראי על הצגת כל הcommits שנעשו בפרויקט בbranch מסוים/כל הcommits שנעשו בpr מסוים
12. diff: מסך שאחראי על הצגת שינויים בקוד

Encryption (Client)

המחלקה אחראית על כל מה שקשור להצפנה בצד הלקוח

|  |  |
| --- | --- |
| משמעות | שם המשתנה |
| Secret key generated only for client | secret\_key: int |
| Encryption key combined from client, server and public keys | encryption\_key: int | null |
| Fernet object used for encrypting | fernet: Fernet | null |

|  |  |  |
| --- | --- | --- |
| טענת יציאה | טענת כניסה | שם הפעולה |
| Calculates encryption key based on B key | self, server\_mixed\_key: int | set\_encryption\_key |
| Initializes p and g | self, p: int, g: int | set\_p\_and\_g |
| bool finished encryption | self | finished\_encryption |
| Uses p g and secret b to calculate B | self | \_calculate\_mixed\_key |

סקירת חולשות

**שכבת האפליקציה:**

- sql injection:

הסכנה: הכנסת תוכן זדוני לפעולות ה SQL שיגרום להן לפעול באופן לא

מתוכנן

הפתרון: הכנסת קלט מן המשתמש בצורה חכמה שלא תאפשר למשתמש

להריץ פקודות זדוניות, באמצעות חיבור מחרוזות חכם.

- MITM :

הסכנה: תוקף עשוי להשתמש ב sniffer על מנת לגלות את המידע שאני

מעביר ברשת, לראות את ההודעות בנושא הגדרות מצלמה ואת התמונות

ממצלמות האבטחה עצמן.

הפתרון: הצפנת כל המידע שעובר בהודעה (אורך הודעה אינו מוצפן)

באמצעות הצפנה סימטרית, יצירת המפתחות באמצעות פרוטוקול דיפי –

הלמן ובכך אני לא מגלה את המפתח ברשת.

- DOS/DDOS – התחזות:

סכנה: תוקף עשוי ליצור לקוחות מזויפים שיתחזו ללקוחות,

לשלוח דרכן כמות מופרזת של מידע ובכך להאט את המערכת

פתרון: אם אני אקבל התחברויות בקצב מהיר מדי אני אנתק אותן

- גנבת סיסמאות:

הסכנה: הסיסמא של מנהל המערכת תהיה גלויה לכל אדם שיוכל לגשת ל

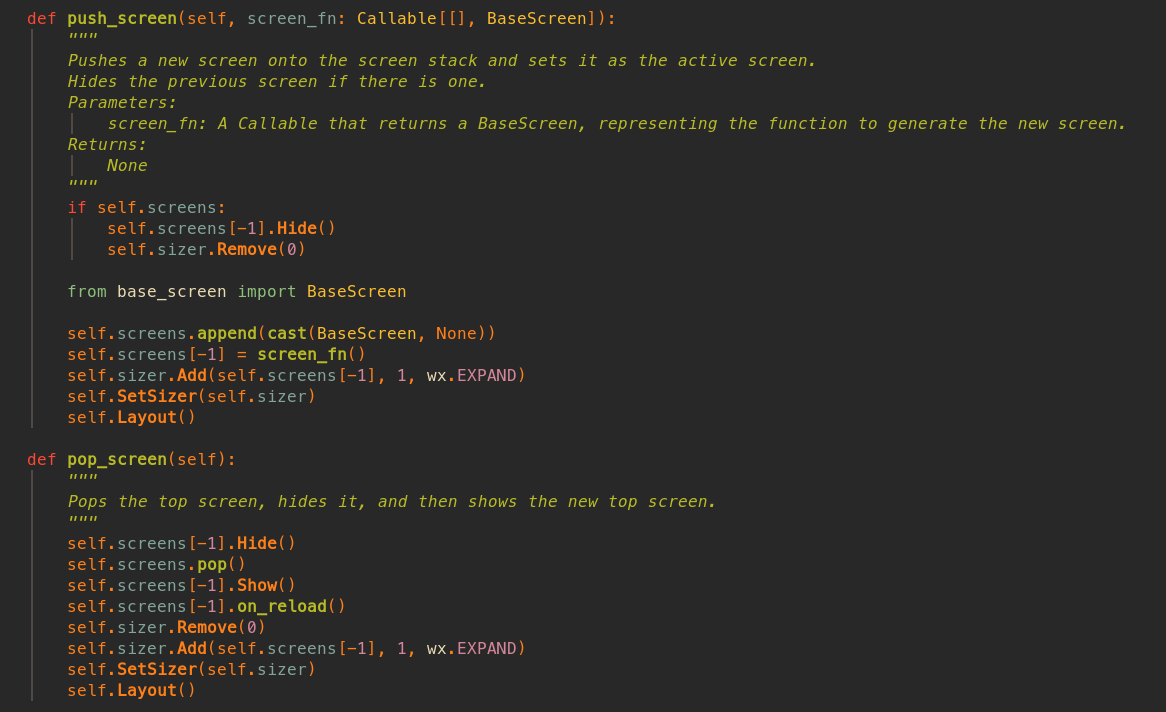
DB

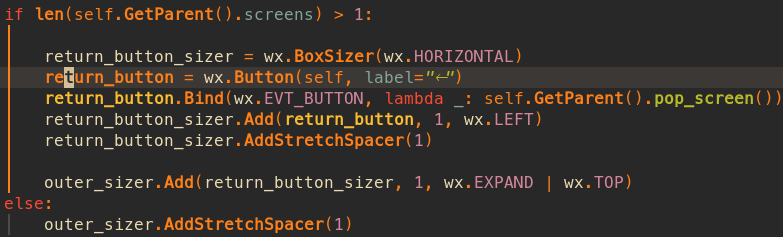
הפתרון: גיבוב סיסמאות (פונקציית hash חד כיוונית) תימנע הבנה של

סיסמאות גם אם מסד הנתונים ייפרץ

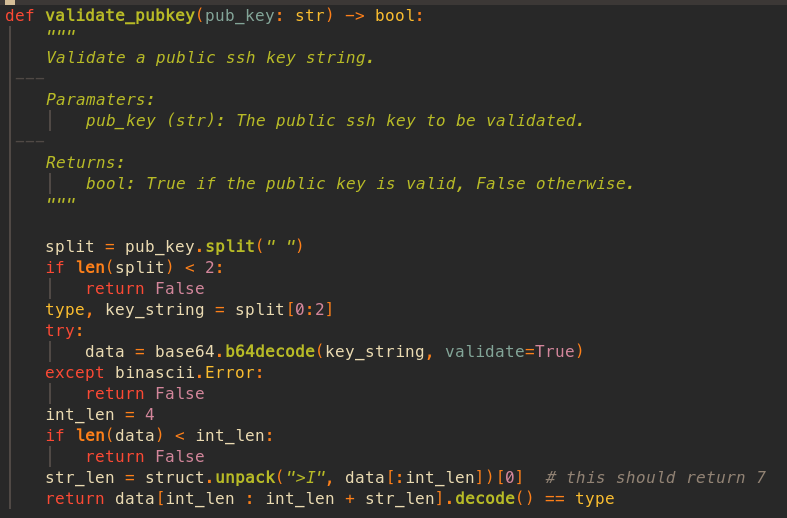
פרק ה’ gitgud הקוד

1. ניהול המסכים של הפרויקט נעשו בעזרת מחסנית ככה שלמשתמש יהיה מאוד קל לחזור למסכים קודמים אחרי שנכנס למסכים אחרים

2. בbase screen אוטומטית נוסף כפתור חזרה אם אתה לא המסך הכי נמוך

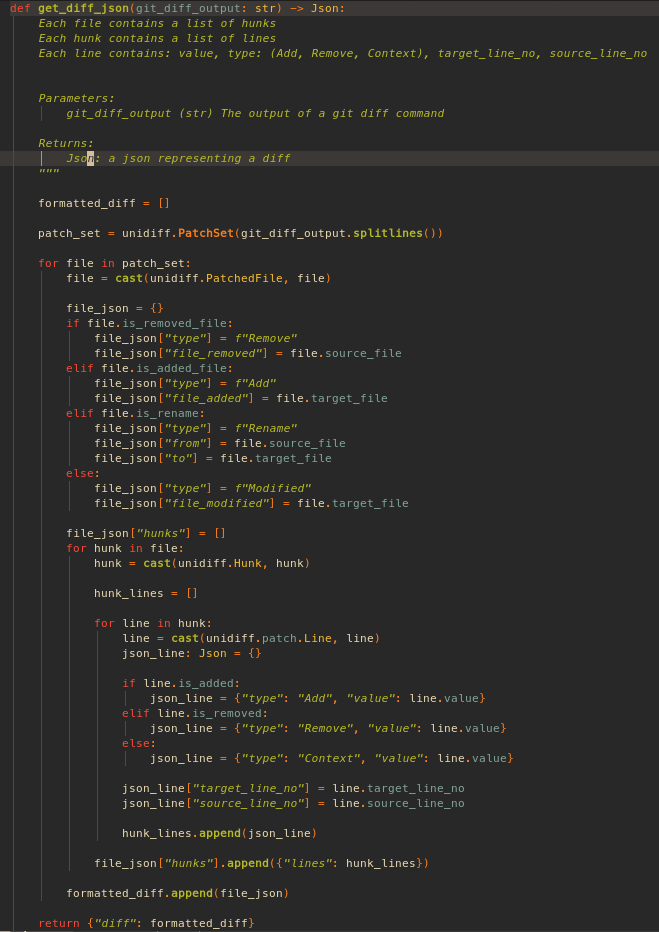


3. לssh יש הגדרה להאם מפתח ציבורי הוא תקין אז בצד השרת מימשתי אותו

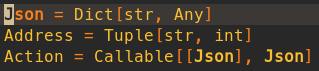
4. בפרוטוקול לכל בקשה יש שם ולכל פעולה יש ורשימה של שדות שהיא חייבת . לכן בהגדרת הפעולות אני מגדיר את רשימת השדות שאני מצפה ובכל בקשה נעשת בדיקת תקינות לפי זה



5. כשאני שולח diff אני שולח אותו כjson בשביל שהגרפיקה תוכל להציג אותו בצורה שהיא תרצה עם צבעים לכל דבר



6. בכל מקום בקוד דאגתי להשתמש ב types והגדרתי כמה types ששימשו אותי במקומות רבים בקוד. הגדרתי את הtypes האלה כי כשקוראים את הקוד קל יותר להבין את המשמעות של דברים כשיש type aliases



7. השתמשתי בספריה בשם fuzzy wuzzy בשביל החיפוש בפרויקט שלי. הספריה מאפשרת חיפוש בטקסט ויודעת להגיד שדברים דומים גם כשיש שגיאות כתיב בגלל שיש בתוכה אלגוריתם השוואת טקסט



הסבר ליצירת סביבת עבודה של השרת

הקוד שלי והcontainer של gitolite נועדו לרוץ בלינוקס ולכן חייב להריץ את הפרויקט בלינוקס

1. התקינו docker מ <https://www.docker.com/>
2. עקבו אחרי ההוראות באתר זה (שימו לב שהpath של מפתח הssh תקין) https://github.com/jgiannuzzi/docker-gitolite?tab=readme-ov-file
3. התקינו את הcontainer של postgres כאן <https://hub.docker.com/_/postgres> השתמשו בשם gitgud-postgres
4. תריצו את הפקדוה הבאה: docker exec -it gitgud-postgres psql -U postgres ואז בתוך psql תכתבו: CREATE DATABASE gitgud
5. תתקינו את הפרויקט ותריצו בעזרת הקובץ שבתוכו את הפקודה : cat ./schema.sql | docker exec -i gitgud-postgres psql -U postgres -d gitgud
6. תתקינו את הספריות של הפייתון שבrequirements.txt של צד השרת
7. יצרו בתקיה server קובץ בשם .env והדביקו בתוכו את הקונפיגורציה הבאה:

https://stackoverflow.com/questions/17157721/how-to-get-a-docker-containers-ip-address-from-the-host

DB\_IP=""

DB\_PASSWORD=""

DB\_NAME=""

DB\_USER=""

SERVER\_PORT=""

צד לקוח

1. התקינו את הקוד
2. התקינו את הספריות ב requirements.txt
3. יצרו בתקיה client את בקובץ .env והכניסו אליו את הקונפיגורציה הבאה:

SERVER\_PORT=""

SERVER\_IP=""