**ספר פרויקט - ענן**



**שם: אילן ישוק**

**ת"ז: 326102126**

**תאריך הגשה: 15 ביוני, 2022**

**מנחה: ברכה עסיס**

**חלופה: תכנון ותכנות מערכות מידע**

**שם הפרויקט: Safe Cloud**

**בי"ס: מקיף אמי"ת באר שבע**

**תוכן עניינים**

**מבוא**

**תיאור תכולת הפרויקט**

מסמך זה מאפיין, מפרט ומציג את התוצר הסופי אותו אני מגיש במסגרת פרויקט הגמר במגמת הנדסת תוכנה בכיתה י"ב. בנוסף לתוצר וליכולות שלו, הוא מציג אתגרים, דרכי התמודדות עמם, הסברים לדברים עקיפים בנושא ועוד מגוון רב של תכנים. מסמך זה מהווה את כל מה שצריך לדעת על הפרויקט, ויותר מכך.

**מוטיבציה לפרויקט**

כולם מכירים את גוגל דרייב. זו אחת מהתוכנות המשומשות בעולם. כל כך משומשת, עד שהיא שינתה את הדרך שבה אנו מעבירים קבצים ממחשב למחשב. אמנם גוגל דרייב מצויין בכל מידותיו, אבל אני תמיד רציתי משהו פשוט יותר, מהיר יותר, שניתן לבצע פעולות כמו הוספת/מחיקת/הורדת קבצים בלחיצה אחת.

אני לא מתיימר להיות גוגל דרייב או איי-קלאוד הבא ולא מנסה להתחרות בעשרות אלפי אנשים, אבל אני כן מציג דרך נוספת למשתמש, פשוטה ומהירה לשמירת קבצים בענן, בצורה הכי מאובטחת שיש (אם ירצה בזאת).

**הפרויקט**

הפרויקט עצמו הוא ענן, ששומר את הקבצים שלך במאגר נתונים הרחק ממך, ובצורה מאובטחת אם תבחר בכך, בשביל שתוכל להשתמש בקבצים האלו במקומות אחרים, במחשבים אחרים ובזמנים אחרים. אני ניסיתי בכוונה לעשות את הממשק הגרפי פשוט ומהיר ככל הניתן, אני לא אוהב להתעסק עם כל הפונקציונליות (שלדעתי מיותרת) בהרבה מהעננים שיש כיום ולכן ניסיתי לצמצם ככל הניתן במישור הזה.

בשביל להשתמש בענן אתה צריך משתמש. לאחר הרשמה שכוללת אימות דו-שלבי עם האימייל, אתה תגיע לאתר שיכיל את הקבצים שלך. אתה תוכל להוסיף קבצים ע"י גרירתם למסך, למחוק ולהוריד בלחיצה. ניתן להוסיף כמות רבה של קבצים בגרירה אחת. בנוסף, אתה יכול לבחור להוסיף ולהוריד קבצים בעזרת מפתח מיוחד. על המפתח אפרט בהמשך, אבל אציין שבעת הכנסת המפתח, המשתמש מוסיף עוד שכבת הגנה – קריטית וטובה מאוד – לקבציו המועברים באינטרנט אל הענן. חוץ מעמוד הקבצים, יש עמוד פרופיל שמציג את פרטיך ונותן לך להתנתק מהמשתמש אם תרצה ועמוד הסבר על הפונקציונליות של התוכנה.

השרת שלי זוכר משתמשים נכנסים ויוצאים, בצורה בטוחה, אצלו בדיסק. תודות לכך, משתמש לא צריך להכנס כל פעם מחדש לאתר, ולמלא את פרטיו. האתר זוכר אותו למשך זמן של יום אחד (שיש שיגידו קצר ויש שיגידו ארוך, אני מאמין שיום אחד זה כמות זמן טובה לנוחות וטובה לאבטחה). כמובן שהשרת לא יודע לא את סיסמת המשתמש, ואם המשתמש יבחר – גם לא את קבצי המשתמש.

**אתגרים שפגשתי במהלך הפרויקט**

פגשתי מס' לא מבוטל של אתגרים בזמן עשיית הפרויקט והנה כמה מהם:

- **אין זמן**. כיתה י"ב עמוסה גם ככה, בגרויות מפיזיקה, מתמטיקה, תנ"ך ולכן היה צריך לנהל את הזמן בצורה יעילה, יחד עם הצבת יעדים לסיום חלקים בעבודה.

- **ממשק גרפי**. כמעט ואף פעם לא עבדתי עם ממשק גרפי כמו שעבדתי בעבודה זו. הממשק הוא אתר אינטרנט, ולכן יצא לי לעבוד עם javascript הרבה, הרבה יותר ממה שידעתי או הכרתי. למדתי מהבסיס ביותר - כיצד javascript רץ ועד מטרותיו והדברים הקטנים והנוחים שהופכים את השפה הזו לכל כך משומשת בעולם.

- **תכנות אסינכרוני** - הרבה בעיות נוצרו עקב זה שjavascript היא שפה שעובדת רק בצורה אסינכרונית. כלומר, היה צריך לשנות את התפיסה שלי - שכל שורה שאני כותב, אני צריך להוסיף לה תכנותית, "לחכות" שהשורה תסתיים ורק אז לשלוח את זה. זה כולל שימוש בPromiseים, async וכו'.

- **אבטחה**. ניסיתי לשים דגש על האבטחה משתי סיבות. ראשית, זהו פרויקט בסייבר, כלומר צריך לגעת באבטחת מידע ורשתות. שנית, בן אדם שירצה לשלוח את הקבצים האלו, ירצה ברוב המקרים, שהם יישארו רק שלו ואף אחד, כולל אני (או מנהל מאגר הנתונים) לא ידע מה יש שם. האתגר הוא - כיצד לעשות את זה? מאיפה מתחילים בכלל?

- **הקלאסי**. לשבת על בעיה כמה שעות, שאין לך שמץ של מושג למה משהו לא עובד בכלל. אתן דוגמה על מה אני מדבר. כשאני מוסיף משתמש למאגר נתונים, אני עושה גיבוב של חלק מהנתונים שלו, בצורה כזו שיהיה קשה מאוד (כל כך שאפשר להגיד בלתי אפשרי) לחזור לנתוניו המקוריים (מהמחרוזת המגובבת). הבעיה שלי הייתה שמשום מה זה לא היה קבוע. בסופו של דבר, אחרי המון בדיקות הגעתי לפתרון והוא להוסיף סוגריים ל datetime.timestamp משום שזו פעולה. נשמע טיפשי נכון? אבל אז קפצה בעיה אחרת. עדיין הגיבוב לא היה קבוע. הפעם גיליתי, שאני שומר תאריך יצירת משתמש מדויק - עד לשניות במאגר הנתונים אבל כשהייתי עושה את הגיבוב (לשם השוואה לאחר הכנסת משתמש נכנס), התאריך היה מדויק לרמת המיקרו-שניות, ולכן בעת הוצאה ממאגר הנתונים וגיבוב המידע, לא הייתי מקבל שווי בין נתונים למרות שהיו שווים. בקיצור.... הקלאסי. וזה קרה פעמים רבות, אבל זה התהליך לתוצר הסופי וכמות הלמידה שניתן להוציא מנסיונות כאלו היא ללא הגבלה.

**קהל יעד**

קהל היעד של הפרויקט הוא בערך כל מי שמשתמש היום באינטרנט - כלומר כל אחד שרוצה את הקבצים שלו ביותר ממקום אחד. אנשים שעובדים עם חומר וצריכים אותו מסודר - רופאים, מורים, הייטק, פקידים, תלמידים ועוד...

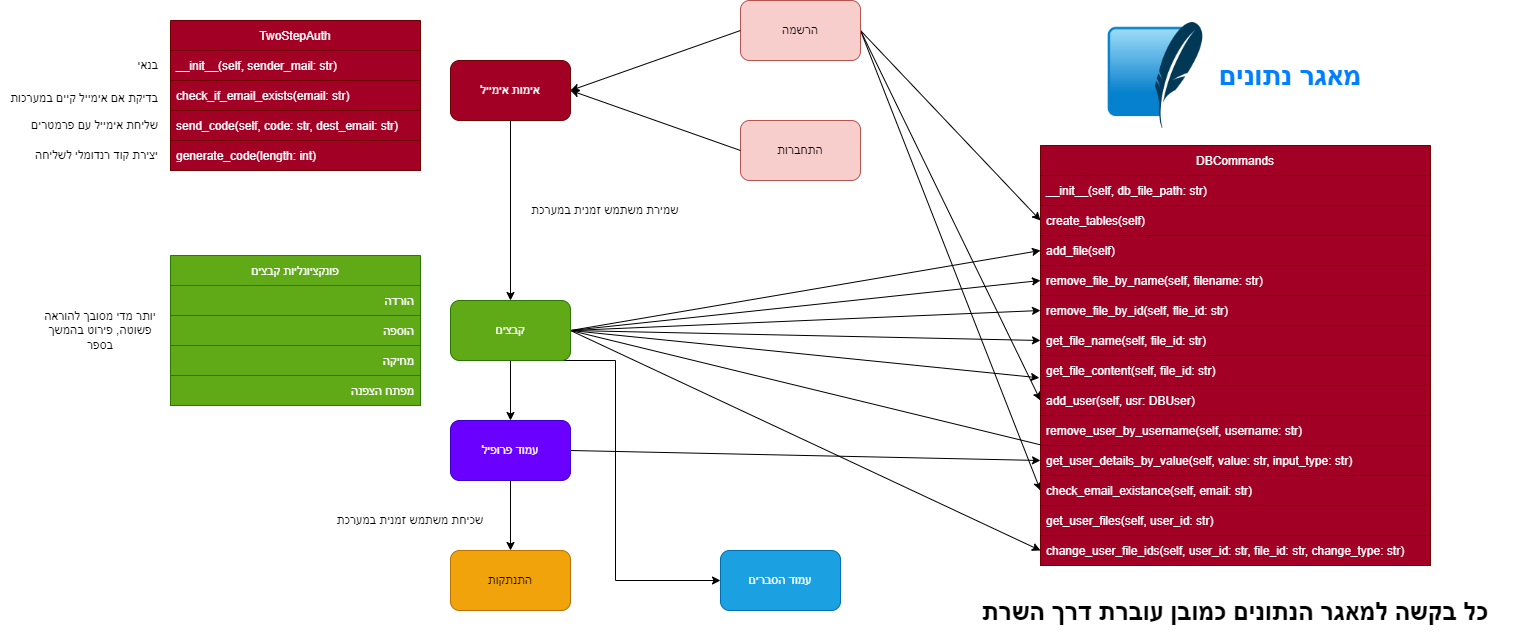
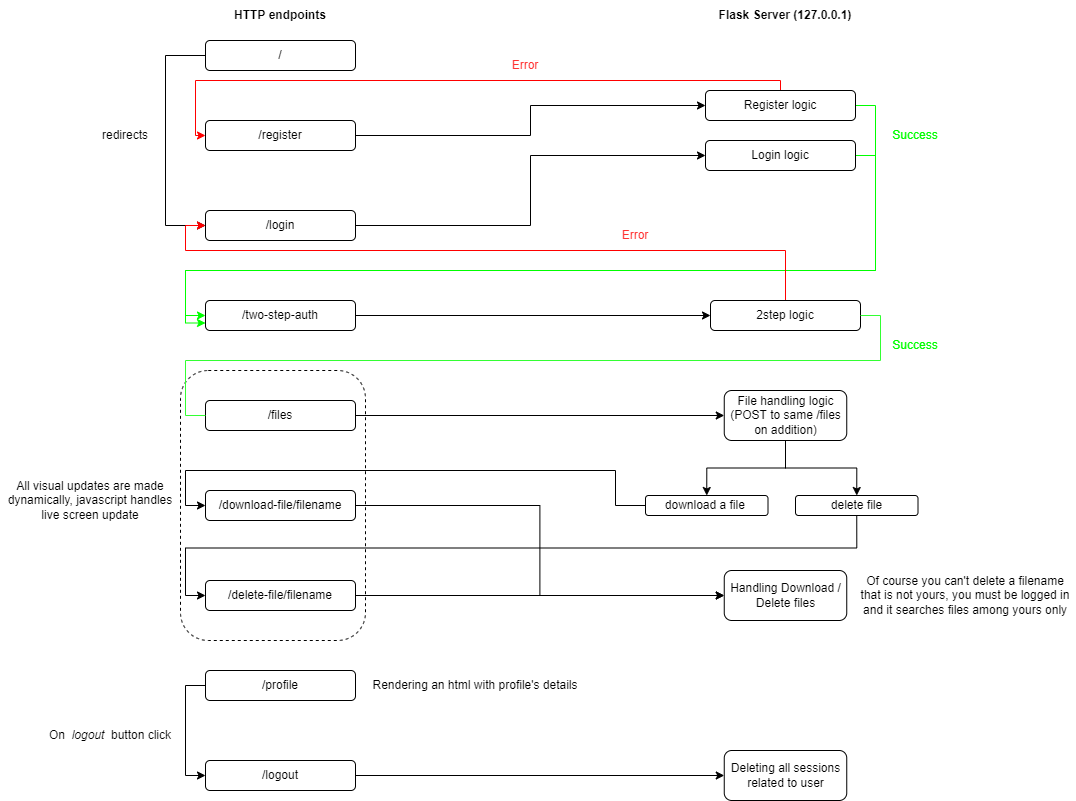
**פיתוחים עתידיים**

כמעט כל מה שתכננתי לפרויקט, בוצע. לכן, אין הרבה פערים שנוצרו אבל יש דברים שהיה נחמד להוסיף:

- שיתוף קבצים בין כמה אנשים, כלומר קישור קובץ מסוים ליותר מבן אדם אחד

- יצירת תיקיות שיכילו קבצים (כמובן לא באופן "זריקת תיקייה"... כי בדקתי... לא כל כך אפשרי)

**ארכיטקטורת הפרויקט**

****ניתן לייצג את ארכיטקטורת הפרויקט באמצעות התרשימים הבאים:

כמו כן, לחלק את הפרויקט לכמה מישורי עבודה שונים:

- **מאגר נתונים**:

- טבלת משתמשים

- טבלת קבצים

- **אבטחה**:

- אימות משתמש דו-שלבי

- הצפנת קבצים במאגר הנתונים

- גיבוב פרטיו של המשתמש ליצירת מחרוזת מזהה ייחודית

- **ממשק גרפי**

- תהליך הרשמות משתמש חדש

- תהליך התחברות של משתמש קיים

- ניהול קבצים (הוספה/מחיקה/הורדה)

הממשק הגרפי הוא אתר, ולכן בקשות HTTP (POST, GET) מהאתר יהיו הקלטים. הבקשות יהיו הקלטים לשרת והכנסת הנתונים לאתר עצמו יהיו הקלטים של המשתמש. קלטים של שאר המישורים, אבטחה ומאגר נתונים, אלו קלטים שמספק אך ורק השרת (כמובן לפי בקשות המשתמש).

**הסדר שמוצג פה הוא גם הסדר לפיו יוצגו החומרים בספר הפרויקט, משום שבדרך זו הכי קל להבין מדוע כל דבר נעשה בצורה שנעשה. הסבר מאגר הנתונים יהיה תחת הכותרת 'בסיס נתונים' לאחר המדריך למשתמש.**

**ארכיטקטורת רשת – שרת לקוח (Flask)**

ארכיטקטורת הרשת היא שרת-לקוח, ארכיטקטורה פופולרית מאוד לשם תוכנות כמו שאני יצרתי. כלומר, בקשות חד-פעמיות מהלקוח אל השרת בכל פעם שהוא צריך משהו. לדוגמה, בכניסה לדף הרשמות, הלקוח (או יש שיגידו הדפדפן) ישלח בקשה לכתובת <http://127.0.0.1:5000/login> ויקבל תשובה בהתאם למה שיבקש. אם ישלח בקשת [HTTP GET](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/GET) יקבל את הדף של התחברות, אם ישלח בקשה [HTTP POST](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/POST) עם מידע מסוים, הוא יקבל עליו פידבק מסוים מהשרת, ואם הפידבק חיובי (ההתחברות צלחה) אז הוא יופנה ל <http://127.0.0.1:5000/two-step-auth>.

לפני שאציג את הפעולות השונות שמתארות את **זרימת התוכנה**, אני אראה את המשתנים שיהיו לנו רלוונטיים בהמשך, להבנה טובה יותר של הפעולות.

אתחיל מהספריות שאני מייבא לקובץ

*from* base64 *import* b64encode

*from* datetime *import* datetime, timedelta

*from* flask *import* Flask, jsonify, session, redirect, url\_for, render\_template, request *as* rq

*from* flask\_session *import* Session *# Server-side session*

*from* CloudServer *import* CloudServer

*from* TwoStepAuth *import* TwoStepAuth

*import* CloudEncrypt

*import* re, os

הספריות כוללות:

* קידוד בbase64
* זמנים בפייתון
* Flask, שזה הנס שבזכותו הכל מקושר בין אתר אינטרנט לבין הקוד. זוהי מעטפת, שבזכותה ניתן ליצור שרת שיטפל בבקשות HTTP לכתובת מסוימת שאנחנו נחליט.
* Sessions, הדרך לשמור נתונים על המשתמש, לאורך כל ריצת הקוד, והדפים השונים. **יש לשים לב** שאני מייבא sessions פעמיים משום שאני רוצה ששמירת הנתונים תהיה בתוך מערכת הקבצים של השרת. אני רוצה את זה מטעמי בטיחות, אסביר בהמשך.
* CloudServer, TwoStepAuth, CloudEncrypt, ספריות שאני כתבתי. הראשונה מתעסקת בגיבוב נתונים, השנייה באימות דו-שלבי והשלישית מטפלת בבקשות שונות של המשתמש.
* Re, דרך לבדוק שהמשתמש מכניס מחרוזות בתבנית מסוימת שאנחנו קובעים
* Os, ספרייה שמתארת את מערכת ההפעלה. מיועדת לפעולות שרק מערכת ההפעלה יכולה לבצע.

משתנים שאני שומר לפני הרצת הקוד:

DB\_FILENAME = os.path.abspath('./database.db')

EXTENSION\_ICONS\_PATH = os.path.abspath('./images/extension\_icons') + '/'

tsa = TwoStepAuth('ilan147963@gmail.com')

auth\_codes = [] *# list of {email: code, email: code} used for 2step auth*

*# just to prevent it from be shared in session*

app = Flask(\_\_name\_\_)

app.config['SESSION\_TYPE'] = 'filesystem' *# For server-side sessions*

Session(app)

*# Secret key, session encrypting*

app.secret\_key = 'dev' *# Const in development*

app.config.from\_pyfile('config.py', *silent*=True) *# Overrides if config.py exists*

app.permanent\_session\_lifetime = timedelta(*days*=1)

cs = CloudServer(DB\_FILENAME, EXTENSION\_ICONS\_PATH)

* DB\_FILENAME, כתובת הקובץ של מאגר הנתונים
* EXTENSION\_ICONS\_PATH, כתובת התיקייה של האייקונים
* Tsa, מחלקת אימות דו-שלבי
* Auth\_codes, משתנה שמקשר בין כתובות אימייל לקודים. ככה אנו זוכרים איזה קוד אנו יוצרים לאיזה אימייל באימות הדו-שלבי
* מגדירים את המפתח הסודי שמשמש להצפנת הsessions
* מגדירים שהsession ישכח לאחר יום אחד
* יוצרים מחלקה של השרת (CloudServer), שאחראית לפעולות שיבקש השרת

כעת אנחנו מוכנים לצאת לדרך! אנסה להציג בעיקר את הדברים החשובים, משום שאם אציג הכל אורך הספר יהיה ארוך בצורה יוצאת דופן. להלן הפעולות מצד השרת שמופעלות לפי בקשות HTTP:

הפעולה הכי בסיסית, מפנה אדם שפונה לכתובת יחסית '/' אל דף התחברות:

@app.route('/')

def home():

*if* 'user\_id' in session:

*return* redirect(url\_for('files'))

*return* redirect(url\_for('login'))

**פעולת רישום (register)**

משתמש חדש, כאשר אדם מופנה ליצירת משתמש:

@app.route('/register', *methods*=['POST', 'GET'])

def register():

*# The weird microsec cutting causes problems in login so I cut it*

אם המשתמש יטען את האתר הוא יקבל את עמוד ההרשמות, אם יש שגיאה היא תוצג.

    err\_msg = session.pop('err\_msg', None)

*if* err\_msg:

*return* render\_template('register.html', *err\_msg*=err\_msg)

*return* render\_template('register.html')

אם המשתמש ישלח נתונים להרשמות משתמש. השרת יבדוק האם אפשר לרשום את המשתמש (יבדוק שאימייל או משתמש לא תפוסים). אם הכל יתבצע כראוי, השרת יסמן שהאדם הספציפי הזה עתיד להירשם (לשימושים עתידיים) וישלח אותו לעמוד אימות האימייל.

*if* rq.method == 'POST':

        email = rq.form['email']

        username = rq.form['username']

        password = rq.form['password']

        ans = cs.register\_possibility(email, username)

*if* ans['code'] == 'success':

*# 2step auth is unique for both login, signup*

*# But session vars preperation is needed*

            session['password'] = password

            session['username'] = username

            session['email'] = email

            session['action'] = 'register'

*return* redirect(url\_for('two\_step\_auth'))

*else*:

*return* render\_template('register.html', *err\_msg*=ans['msg'])

**פעולת חיבור משתמש קיים (login)**

@app.route('/login', *methods*=['POST', 'GET'])

def login():

למקרה של בקשת טעינה:

    err\_msg = session.pop('err\_msg', None)

*if* err\_msg:

*return* render\_template('login.html', *err\_msg*=err\_msg)

*return* render\_template('login.html')

למקרה של שליחת נתונים:

*if* rq.method == 'POST':

        usr\_input = rq.form['user\_input']

        usr\_pass = rq.form['password']

        ans = cs.try\_login(usr\_input, usr\_pass)

*if* ans['code'] == 'error':

*return* render\_template('login.html', *err\_msg*=ans['msg'])

*else*:

            user = cs.get\_user\_details(ans['msg'], 'user\_id')

*if* not user:

*return* render\_template('login.html', *err\_msg*='No such user..')

            session['email'] = user.email  *# Where to send*

*return* redirect(url\_for('two\_step\_auth'))

הפעולה בודקת האם אפשר להכניס את המשתמש לפי הנתונים שהכניס. מאחר ואפשר להכניס אימייל ושם משתמש בעמודה העליונה בהתחברות, הפעולה try\_login אינה פשוטה, אתם מוזמנים להסתכל עליה במדריך למפתח. לאחר הבדיקה, אם יש בעיה בהתחברות אנו מציגים אותה, ואם אין – מפנים לחלון לעמוד אימות האימייל.

**פעולת אימות אימייל (two\_step\_auth)**

הפעולה הזו יותר מורכבת מכל מה שנכתב עליו עד כה, משום שהיא פעולה שפועלת גם על משתמש מתחבר וגם על משתמש חדש שנרשם, אז יש צורך לתמרן בין השניים.

לפני הפעולה, נגדיר משתנים שיעזרו לנו בהמשך:

tsa = TwoStepAuth('ilan147963@gmail.com')

auth\_codes = [] *# list of {email: code, email: code} used for 2step auth*

*# just to prevent it from be shared in session*

הכותרת:

@app.route('/two-step-auth', *methods*=['POST', 'GET'])

def two\_step\_auth():

    redirect\_to = 'files' *# Where after 2step auth succeeds*

העמוד הבא אליו נפנה תלוי במשתנה redirect\_to, כרגע זהו עמוד הקבצים, שאנו מפנים אליו אחרי התחברות תקינה.

למקרה של טעינת העמוד:

*if* 'email' not in session:

*return* redirect(url\_for('login'))

    email = session['email']

*if* not tsa.check\_if\_email\_exists(email):

        session['err\_msg'] = 'Email does not exist!'

        session.pop('email', None) *# Clear session*

*return* redirect(rq.referrer)

    code = tsa.generate\_code(6)

    auth\_codes.append({'email': email, 'code': code})

    tsa.send\_code(code, email)

*return* render\_template('code\_enter.html')

אנו בודקים האם המשתמש הכניס אימייל. אם לא – מפנים חזרה לחיבור ואם כן, בודקים שהאימייל לא רק בעל תבנית נכונה אלא רשום בתיבות השונות. אם האימייל לא מאושר, אנו מחזירים את המשתמש לפעולתו הקודמת. אם כן, אנו יוצרים קוד, מוסיפים אותו לטבלה גלובלית (auth\_codes) שמקשרת בין אימייל לקודים (בשביל לעקוב אחרי הקודים שאנחנו שולחים, שנוכל להשוות). לבסוף, אנו שולחים את הקוד לתיבת הדואר של המשתמש.

למקרה של הכנסת נתונים. אנו קוראים את מה שהוכנס, ומשיגים את האימייל והפעולה שהמשתמש עושה. הפעולה היא רישום/התחברות.

*if* rq.method == 'POST':

*# If we received a POST req, then we already*

*# did a GET here, and values are popped*

*# so this if must be first, to not get redirected*

        received\_code = rq.form['code\_input']

        email = session.pop('email', None)

        user\_action = session.pop('action', None)

עכשיו, נבדוק האם יש התאמה בין הקוד שנשלח ושמור בauth\_codes לבין הקוד שהוכנס. אם יש התאמה, נגדיר את זכירת המשתמש לקבועה (למשך יום אחד)

*for* ac *in* auth\_codes:

*if* ac['email'] == email and ac['code'] == received\_code:

*# code that is typed is connected to the email*

*# success*

                auth\_codes.remove(ac)

                session.permanent = True

*# Only in register -> Create an account*

*if* user\_action == 'register':

                    print('registering')

                    password = session.pop('password')

                    username = session.pop('username')

*# Convertion is to delete micro-secs from now(), makes problems...*

                    creation\_time = datetime.now().strftime("%d/%m/%Y, %H:%M:%S")

                    creation\_time = datetime.strptime(creation\_time, "%d/%m/%Y, %H:%M:%S")

                    user\_id = CloudEncrypt.hashify\_user(email, password, creation\_time)

                    cs.register(user\_id, username, email, creation\_time)

                    session['user\_id'] = user\_id *# as a setup to /files*

אם המשתמש מנסה להרשם, אנו יוצרים לו משתמש, מגבבים לו את הנתונים בצורה שתיארתי מקודם בקובץ תחת הכותרת 'אבטחה' ולבסוף, קוראים לפעולה register שרושמת את המשתמש במאגר הנתונים. אנו גם מוסיפים לsession שהוא אחראי על שמירת משתנים לטווח הקצר את מס' הזיהוי של המשתמש בשביל שנוכל לתקשר אתו בצורה בטוחה אחר כך.

אם המשתמש מנסה להתחבר ולא להרשם, נדלג על כל השלב של הרישום שמתרחש למעלה, ופשוט נעביר אותו לעמוד הקבצים:

*else*:

                    session['user\_id'] = cs.get\_user\_details(email, 'email').user\_id *# as a setup to /files*

                session['filenames'] = []

*return* redirect(url\_for(redirect\_to))

בכל הפעולות הבאות אנחנו בודקים תחילה אם המשתמש מחובר, משום שאם לא – אסור שיהיה לא את היכלות להתחבר בכלל. אנו בודקים זאת באמצעות חיפוש זהותו בsession, אם יש לו מס' זיהוי רשום, אז הוא יכול להתחבר. אם אין לו, אז לא.

**פעולת הקבצים** **(files)**

אם יש משתמש מחובר, אנחנו משיגים את קבציו באמצעות get\_user\_filenames ו-get\_file\_icon\_content (מידע לתמונת סיומת הקובץ ומידע על שמות הקבצים). אנו שומרים את שמות הקבצים בsession בשביל שנזכור אותם למשך חיבור המשתמש ולא נצטרך לטעון אותם כל פעם מחדש. אחרי השגת כל הקבצים, אנחנו טוענים אותם למסך בעזרת תבנית שקיימת מראש ע"י פעולת render\_template.

@app.route('/files', *methods*=['POST', 'GET'])

def files():

*if* 'user\_id' not in session:

*return* redirect(url\_for('login'))

*if* rq.method == 'GET':

*# Fetch user's files*

        user\_id = session['user\_id']

        filenames = cs.get\_user\_filenames(user\_id)

        session['filenames'] = filenames

        files\_data = []

*for* fn *in* filenames:

            icon\_content = b'data:image/png;base64,' + cs.get\_file\_icon\_content(fn)

            files\_data.append(

                {'name': fn,

                'icon\_data': icon\_content.decode()})

*return* render\_template('files.html', *files\_data*=files\_data)

אם משתמש רוצה **להוסיף** קובץ, הוא ישלח בקשת POST לאותה כתובת. במקרה זה השרת יאסוף את המידע הרלוונטי על הקבצים שהוכנסו (או קובץ יחיד, עובד לכל המקרים). לאחר מכן, לכל קובץ, הוא ישנה את השם בהתאם להימצאותו במאגר הנתונים (בשביל לא ליצור בעיות שכתוב קבצים).

*if* rq.method == 'POST':

        icons\_data = []

        files\_num = int(rq.form['length'])

*for* i *in* range(files\_num):

            file = rq.files[f'files-{i}']

            name = file.filename

            content = file.stream.read()

*# Manage duplicates, edits session['filenames']*

            saved\_filename = configure\_filename(name, session['filenames'])

לבסוף הוא יוסיף את הקובץ וישייך אותו למשתמש על ידי add\_file\_to\_user. בנוסף, הוא יוסיף את תמונת הסיומת של האייקון ושמו למשתנה מסוג מילון שיישלח לאחר מכן למשתמש.

            file\_id, file\_icon\_data = cs.add\_file(False, saved\_filename, content)

*# Update user's files*

            cs.add\_file\_to\_user(session['user\_id'], file\_id)

            icons\_data.append({'data': file\_icon\_data.decode(), 'filename': saved\_filename})

*return* jsonify(icons\_data)

המשתמש כעת יוכל ליהנות מהתעדכנות אוטומטית של הקבצים אצלו על המסך.

**פעולת הורדת קובץ (download\_file)**

תחילה הפעולה בודקת אם הקובץ שנתבקש להורדה, הוא של המשתמש. אם השם של הקובץ אכן יהיה של המשתמש, אז תוכן הקובץ של המשתמש יוחזר חזרה למבקש הפעולה (לקורא לכתובת '/files/download/some\_filename.txt').

@app.route('/files/download/<filename>')

def download\_file(*filename*: str):

*if* 'user\_id' not in session:

*return* redirect(url\_for('login'))

*# Search for filename among only the logged account*

    files\_id = cs.get\_user\_file\_ids(session['user\_id'])

*for* fid *in* files\_id:

*if* cs.get\_filename(fid) == *filename*:

*# Found relevant file, so I can download it*

            content = cs.return\_file\_content\_by\_id(fid)

*return* b64encode(content)

*return* redirect(url\_for('login')) *# If file wasn't found*

**פעולת מחיקת קובץ (delete\_file)**

דומה לפעולת ההורדה, רק שבמקרה זה אנו מוחקים את הקובץ ומורידים את הבעלות של המשתמש לקובץ. כמו כן, אנו מורידים את שם הקובץ מהsession לטעינה עתידית. כעקרון אנו לא צריכים להחזיר כלום, אבל מפני שזו חובה, אנחנו מחזירים שמחקנו את הקובץ.

@app.route('/files/delete/<filename>')

def delete\_file(*filename*: str):

*if* 'user\_id' not in session:

*return* redirect(url\_for('login'))

*# Search for filename among only the logged account*

    files\_id = cs.get\_user\_file\_ids(session['user\_id'])

*for* fid *in* files\_id:

*if* cs.get\_filename(fid) == *filename*:

            cs.delete\_file\_by\_value(fid, 'file\_id') *# files*

            cs.remove\_file\_from\_user(session['user\_id'], fid) *# user*

            session['filenames'].remove(*filename*) *# session*

*# Doesn't really matter if we didn't delete something that didn't exist*

*return* jsonify({'message': 'Delete occurred'})

*return* redirect(url\_for('login')) *# No file found*

**פעולת הפרופיל (profile)**

פעולה פשוטה מאוד שבעיקרה מציגה את הפרופיל ונותנת אפשרות להתנתק מהאתר למשתמש. משיגה את כל המידע הרלוונטי בעזרת get\_user\_details ואם חוזר משתמש קיים, היא טוענת את המסך עם נתוניו.

@app.route('/profile')

def profile():

*if* 'user\_id' not in session:

*return* redirect(url\_for('login'))

    user = cs.get\_user\_details(session['user\_id'], 'user\_id')

*if* user:

*# Send in it parts because I don't want all the details*

*# out there, sounds risky*

*return* render\_template('profile.html',

*username*=user.username,

*email*=user.email,

*creation\_date* = user.creation\_date.strftime("%d %B, %Y"),

        )

*return* 'error'

**פעולת התנתקות (logout)**

בפנייה לכתובת זו, האתר ישכח את המשתמש. לאחר הניתוק הוא יכוון אותנו חזרה לעמוד ההתחברות.

@app.route('/logout')

def logout():

*if* 'user\_id' in session:

        session.pop('user\_id', None)

        session.pop('filenames', None)

*return* redirect(url\_for('login'))

**אבטחה – גיבוב נתונים, אימות דו-שלבי, הצפנת צד לקוח**

לכל הקבצים הרלוונטיים, ראה כותרת "המדריך למפתח". בחלק זה אציג את הדברים השונים שאני עושה למען ביטוח והצפנה של פרטי ומידע המשתמש. הכותרות מדורגות לפי רמת האבטחה, כלומר גיבוב נתונים מהווה אבטחה ברמה גבוהה, אך אין פה קונספט יוצא מן הכלל. לעומת זאת, הצפנת צד-לקוח היא מההצפנות החזקות שקיימות, לא מעצם אלגוריתם מופלא או טריק מתמטי (למרות שגם מזה אפשר להתפאר) אלא מהקונספט אותו אסביר תחת התת-כותרת המתאימה.

**גיבוב נתונים – המס' זיהוי של המשתמש**

אסור בשום פנים ואופן שהשרת, או האחראי על מבנה הנתונים, יוכל לגשת למשתמשים השמורים במאגר. דבר כזה יהיה אסון אם יתגלה על ידי המשתמשים משום שיהווה פגיעה איומה בפרטיות שלהם. לצורך מניעת בעיה כזו, אני מקבל את פרטי המשתמש, מפעיל עליהם פונקציית גיבוב, שכידוע לא ניתן לשחזרה (מכיוון שמדובר [בפונקציות מתמטיות חד-כיווניות](https://he.wikipedia.org/wiki/%D7%A4%D7%95%D7%A0%D7%A7%D7%A6%D7%99%D7%99%D7%AA_%D7%92%D7%99%D7%91%D7%95%D7%91). הנה הקוד:

def hashify\_user(*email*: str, *password*: str, *creation\_date*: datetime):

*# Some dynamic salting for md5 hash*

    creation\_time\_mixed = str(*creation\_date*.timestamp())[::-2]

    salt = hashlib.md5(*email*.encode()).digest()[::2] + \

    hashlib.md5(creation\_time\_mixed.encode()).digest()[::3]

    encstr = hashlib.md5(*password*.encode() + salt).hexdigest()

*return* encstr

* קבלת אימייל, סיסמה, תאריך יצירה כקלט
* הפיכת זמן יצירה לכמות שניות מאז 1970 ובלבול המחרוזת
* יצירת [מלח](https://infosecscout.com/md5-salt-hash/#:~:text=not%20so%20much%20%E2%80%A6-,What%20is%20MD5%20Salt%20and%20How%20to%20Use%20It%3F,to%20get%20the%20same%20result.), שהוא גיבוב מחרוזת אימייל מבולבלת בתוספת גיבוב המחרוזת המבולבלת של זמן היצירה
* גיבוב המלח בתוספת הסיסמה לקבלת התוצאה הסופית
* התוצאה היא גיבוב הנתונים, תוצאה יחידה במינה לכל משתמש, מס' זיהוי המשתמש

**אימות דו-שלבי**

אימות דו-שלבי קיים בשביל לוודא שגם אם פרטיו של המשתמש נחשפו, עדיין לא יהיה אפשר להכנס לאתר משום שיהיה חייב גם להשיג את פרטי המשתמש לאימייל. כלומר, זה מקשה על הפורץ משום שעליו גם לפרוץ לתיבת הדואר של אותו משתמש, דבר שלא יקרה אם המשתמש לא ייפול למרמות מסוימות.

הוא עובד בצורה הבאה, ישנו אובייקט TwoStepAuth שנוצר עם הרצת קובץ המקור של האתר. האובייקט נוצר פעם אחת, מתחבר לתיבת הדואר של גוגל (משום שהאימייל השולח מאוחסן בשרתים אלו).

    def \_\_init\_\_(*self*, *sender\_email*: str):

*# Personal details*

*self*.sender\_email = *sender\_email*

*self*.smtp\_server = 'smtp.gmail.com'

*# Create .env for these*

*self*.username = config('EMAIL\_USERNAME')

*self*.password = config('EMAIL\_PASSWORD')

        print(f'Trying to log in with {*self*.username}, {*self*.password}... ', *end*='')

*self*.conn = SMTP\_SSL(*self*.smtp\_server, 465)

*self*.conn.set\_debuglevel(False)

*self*.conn.login(*self*.username, *self*.password)

        print('Logged in!')

הוא כולל את הפעולות הבאות:

1. בדיקה האם אימייל באמת קיים במערכות השונות (לא רק אם הוא כתוב בצורה מותרת מבחינת תווים, אורך וכו')

    @staticmethod

    def check\_if\_email\_exists(*email\_address*: str):

        url = 'https://isitarealemail.com/api/email/validate' *# A blessing for developers*

        res = requests.get(url, *params*={'email': *email\_address*})

        status = res.json()['status']

*if* status == 'valid':

*return* True

*return* False

2. יצירת קוד רנדומלי

    @staticmethod

    def generate\_code(*len*: int):

        letters = 'abcdefghijklmnopqrstuvwxyz'

        letters\_big = letters.upper()

        digits = '0123456789'

        final\_str = ''

*for* i *in* range(*len*):

            element = random.choice([letters, letters\_big, digits])

            final\_str += random.choice(element)

*return* final\_str

3. שליחת קוד

def send\_code(*self*, *code*: str, *dest\_email*: str):

*# Beautify*

        text\_subtype = 'html'

        content = f"""\

        <p>Thanks for logging in.</p>

        <p>Code: <b>{*code*}</b></p>

        """

        subject = 'SafeCloud code confiramtion'

*try*:

            msg = MIMEText(content, text\_subtype)

            msg['Subject'] = subject

            msg['From'] = 'SafeCloud'

*self*.conn.sendmail(*self*.sender\_email, *dest\_email*, msg.as\_string())

*except* Exception *as* e:

            print(f'Message failed: {e}')

תחילה יש הגדרה ראשונית של האובייקט, שכוללת התחברות לאימייל עם שם משתמש וסיסמה שנמצאים במשתנים סביבתיים (EMAIL\_USERNAME, EMAIL\_PASSWORD).

כאשר משתמש מתחבר או נרשם למערכת, תחילה יש בדיקת תקינות אימייל ולאחר מכן נוצר קוד רנדומלי ונשלח לאימייל שהתחבר.

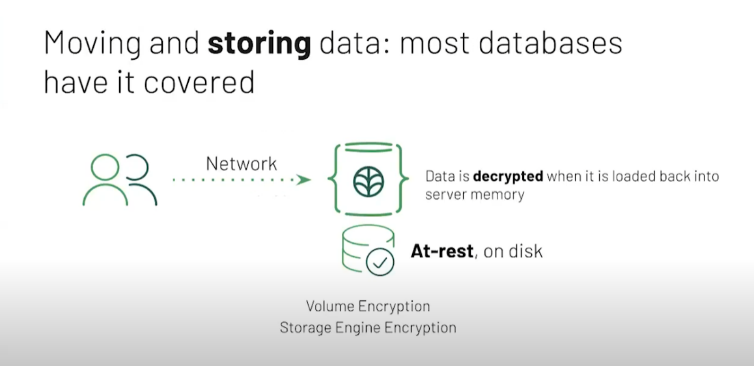
למען ביטחון החיבור, יש סגירה נכונה של החיבור בין תיבת הדואר לאימייל השולח. הסגירה מתבצעת אם הקוד מפסיק לעבוד, או קופצת איזושהי תקלה:

    def \_\_del\_\_(*self*):

        print('Closing smtp connection')

*self*.conn.close()

**הצפנת צד לקוח**

ההצפנה הכי חזקה לדעתי, היא הצפנה שאף אחד מלבדך לא יכול לפצח. הפורץ בעצם יכול לראות את הקובץ שהלקוח שולח, רק ברגע אחד, והוא בין השליחה לבין "המנוחה" במאגר הנתונים. כלומר, אם אני שולח לשרת קובץ, הוא יהיה חשוף לגמרי בינתיים שהשרת עובד עליו, ובזמן זה משהו יכול בצורה כלשהי להאזין לשרת ולגלות את הקובץ (כמובן שאנו מנסים להימנע מכך אך באבטחה צריך תמיד להתכונן למקרה הנורא מכל). בשביל **לטפל** בזה, אנו פשוט מסתירים את הקובץ מהרגע שהוא עוזב את המחשב. ככה, אף אחד, לא השרת ולא מבנה הנתונים ולא הפורץ, אף אחד, לא יוכל לדעת מה יש בקובץ. אנו עושים זאת מאוד פשוט (ולכן אמרתי שהרעיון מלהיב יותר מהביצוע). ההצפנה היא AES שממומשת בספריה CryptoJS בעת העלאת הקובץ (אתמקד בחלקים החשובים), אנו יוצרים פקודה לקריאתו, בצד הלקוח. לאחר הקריאה, אנו מתכוננים לשליחת המידע לשרת אבל לפני זה, בודקים אם הוכנס קוד סודי (ניתן לראות במדריך למשתמש בדף הקבצים). אם כן:

*if* (pk != '') {

*// AES client-side encryption is needed*

          let wordArray = CryptoJS.lib.WordArray.create(file.content);

          let encrypted = CryptoJS.AES.encrypt(wordArray, pk).toString();

          formData.append(`files-${counter}`, new Blob([encrypted]), file.filename);

        }

אנו מצפינים את המידע בקובץ (file.content) ורק אז מוסיפים את המידע שלו. בעת ההורדה אנו עושים אותו דבר, מפענחים לפי מפתח. byteCharacters מייצג בbase64 את הקובץ:

*// Convert base64 to blob*

*if* (pk != '') {

          let decrypted = CryptoJS.AES.decrypt(byteCharacters, pk)

          let typedArray = convertWordArrayToUint8Array(decrypted);

          saveData(filename, `data:${extension}`, new Blob([typedArray]));

*return* false

        }

saveData היא פעולה שדואגת לשמור את הקובץ במחשב המשתמש, ניתן לראות אותה במדריך למפתח.

**שימוש בטכנולוגיות חיצוניות (ו-API)**

להקמת פרויקטים כאלו, לא ניתן לממש הכל מהתחלה מסיבה ראשית ועיקרית, אין למה להמציא את הגלגל מחדש, אם נתמקד כל פעם מחדש בהמצאת הגלגל לעולם לא נגיע למכונית. לכן, השתמשי בטכנולוגיות שונות שאציג כאן:

|  |  |
| --- | --- |
| טכנולוגיה | פירוט |
| [Flask ונגזרותיו](https://flask.palletsprojects.com/en/2.1.x/) | למען הפיכת השרת לשרת HTTP שיכול לטפל בכל בקשות HTTP ולהחזיר מידע בקלות (יחסית לsocket רגיל) |
| [Flask\_session](https://flask-session.readthedocs.io/en/latest/) | בשביל לשמור את המידע הזמני על המשתמשים בקבצי השרת. מטרת הספרייה היא להזיז את מיקום המידע הנשמר מקבצי המשתמש לקבצי השרת, וזאת מטעמי אבטחה – אם אני מעביר פרטים כמו סיסמאות, אני מעדיף שישמרו בשרת. |
| [Hashlib](https://docs.python.org/3/library/hashlib.html) | לגיבוב הנתונים של המשתמשים |
| [Uuid](https://www.techtarget.com/searchapparchitecture/definition/UUID-Universal-Unique-Identifier#:~:text=A%20UUID%20(Universal%20Unique%20Identifier,UUID%20generated%20until%20A.D.%203400.) | ליצור מס' זיהוי ייחודי לקבצים שנוספים למערכת |
| [Base64](https://docs.python.org/3/library/base64.html) | קידוד קבצים בפורמט base64, שהוא הופך מידע לבינארי לטקסטואלי |
| [threading](https://docs.python.org/3/library/threading.html?highlight=threading#module-threading) | דרך בפייתון לעבוד עם תהליכים שונים בו זמנית (לדוגמה לתקשר עם שני משתמשים בו זמנית) |
| [Smtplib](https://docs.python.org/3/library/smtplib.html) | תיקייה שאחרית לחיבור לשרת SSL. השימוש בו הוא לאימות דו-שלבי עם google gmail. בשביל להתחבר לאימייל זה, יש ליצור חיבור מאובטח באמצעות SSL. |
| [Is it a real email?](https://isitarealemail.com/) | API שבודק האם אימייל מסויים הוא אימייל שבאמת רשום במערכת, ולא סתם אימייל שמוקלד |
| [github](https://github.com/) | לשמירת וניהול גרסאות של הקוד. עוזר לסידור, ועבודה בכמה מקומות במקביל. |
| [CryptoJS](https://cryptojs.gitbook.io/docs/) | להצפנת קבצים מצד לקוח בהצפנת AES. משומש בjavascript בקוד לקוח |
| [Ajax / jquery](https://jquery.com/) | ספריות קלות (מבחינת זכרון) המקלות על שליחת בקשות וטעינת גופים בצירה דינמית לתוך אתר. הספריות משומשות בjavascript וכל כך מפורסמות עד שנהיו לסטנדרט |
| [Bootstrap](https://getbootstrap.com/) | ספריית css שעוזרת לעצב דפי אינטרנט בקצב הרבה יותר מהיר מהרגיל. השתמשתי מפני שלא רציתי לבזבז יותר מדי זמן על העיצוב (למרות שבפועל זה גם לא היה פשוט) מאחר וזה לא מטרת הפרויקט |

**מדריך למשתמש**

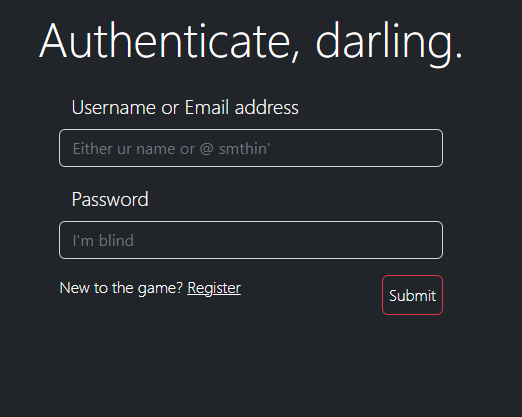
**התקנת התוכנה**

התוכנה נמצאת ב [**virtual enviorment**](https://docs.python.org/3/tutorial/venv.html), כלומר ההתקנה יחסית פשוטה.

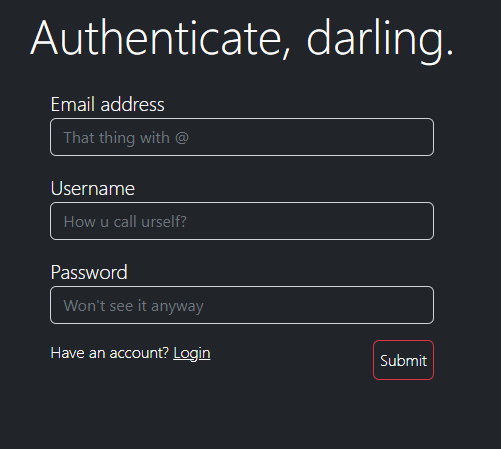
* עושיםgit clone מאתר הפרויקט - <https://github.com/ya5huk/safe_cloud>
* מתקינים את כל התיקיות (שלא צריך לחפש מאחר וזה virtual enviorment)
* בקובץ app.py אפשר לשנות אתDB\_FILENAME בשביל שיהיה קובץ מאגר נתונים אחר.
* יש להגדיר את המשתנים הסביבתיים בעת הרצהEMAIL\_USERNAME, EMAIL\_PASSWORD שייצגו את נתוני האימייל השולח קודים לאימות דו-שלבי
* כעת יש להריץ את הקובץ app.py

**הרצת התוכנה**

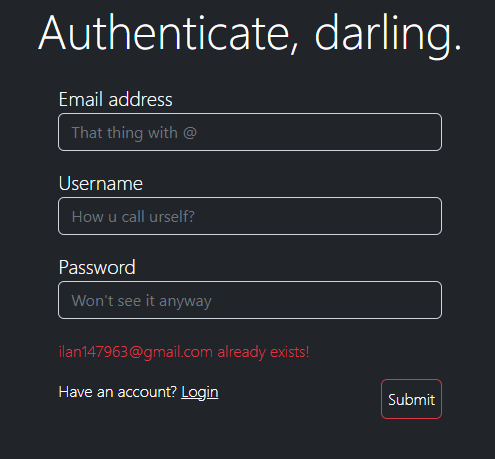
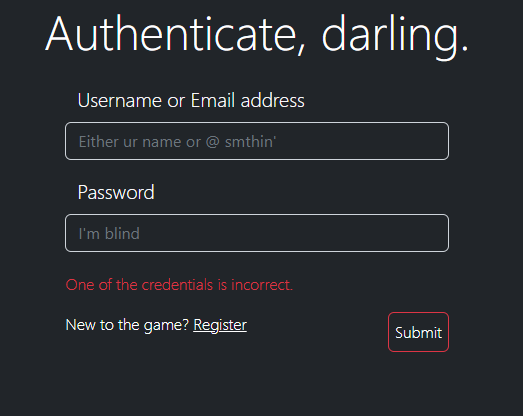
בעת פתיחת האתר שירוץ ב 127.0.0.1:5000 (אלא אם הוגדר אחרת) יפתח **חלון התחברות**.



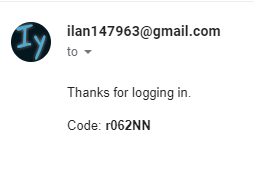
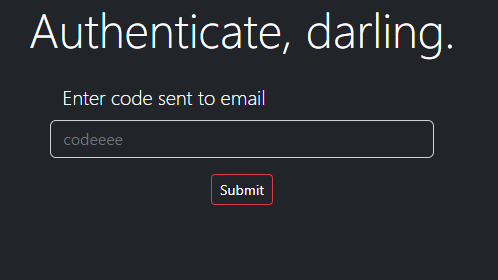
למקרה ואין משתמש, יש ללחוץ על Register בשביל לעשות משתמש חדש. בעת לחיצה **יתקבל חלון רישום משתמש חדש**:



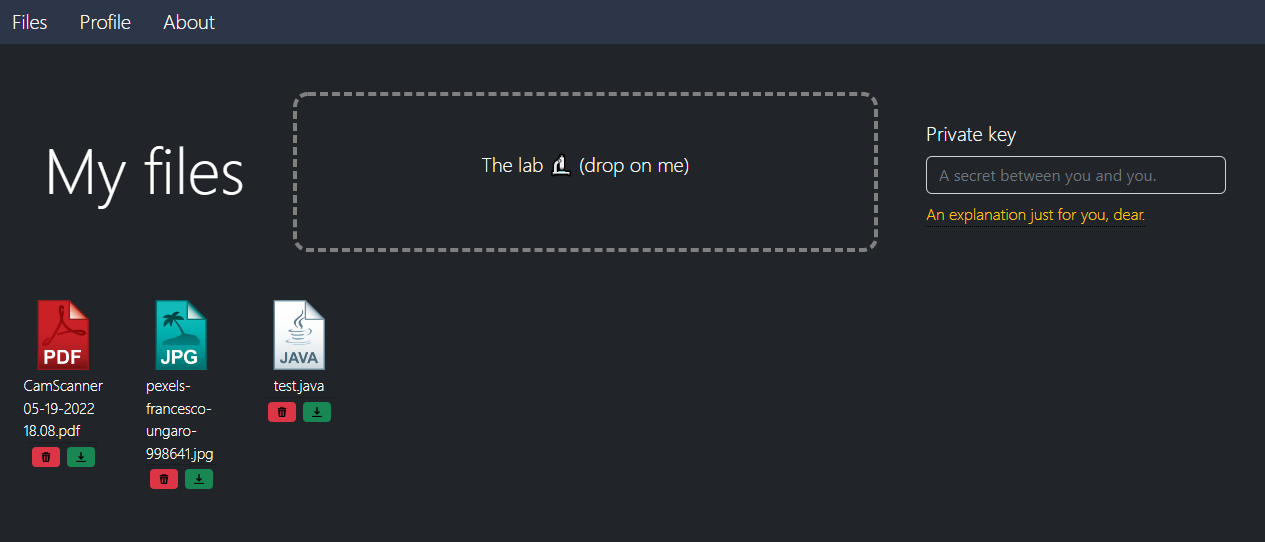
החלונות האלו נותנים פידבק למקרה ויש הכנסת נתונים תפוסה או שגויה:

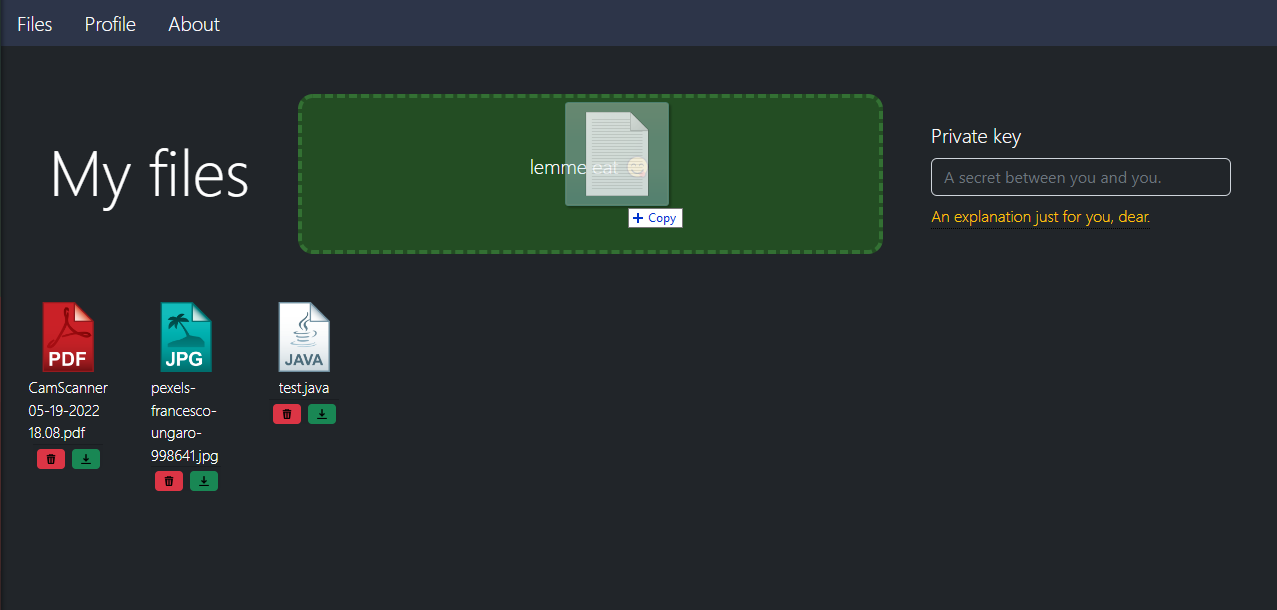
לא משנה מה תהיה הפעולה – רישום משתמש חדש או התחברות, החלון הבא שיקפוץ, לאחר לחיצה על submit הוא **חלון אימות האימייל**. החלון נועד לכך שאנשים לא סתם יכניסו אימייל שהוא לא שלהם, ושלמקרה שגילו את פרטי המשתמש שלהם, עדיין לא יוכלו להכנס למשתמש כי הפורצים לא יחזיקו בתיבת הדואר (שמאובטחת על ידי google, apple, yahoo וכו').

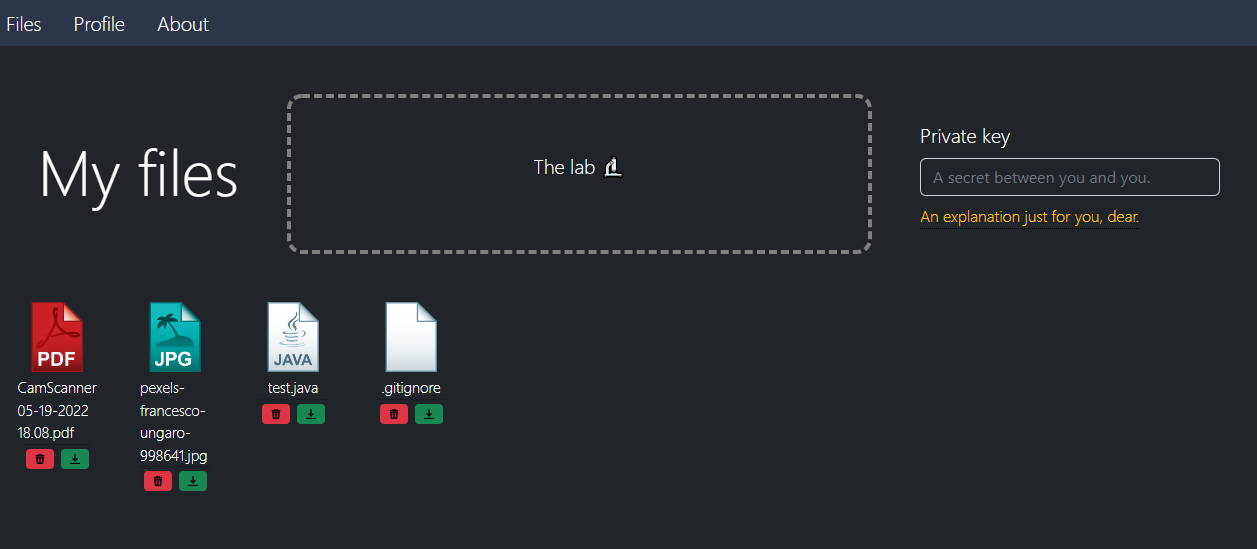
לאחר התחברות תקינה (אם תכניס קוד שגוי, תקבל פידבק על זה), מגיעים לדף העיקרי של הפרויקט והוא **דף הקבצים:**

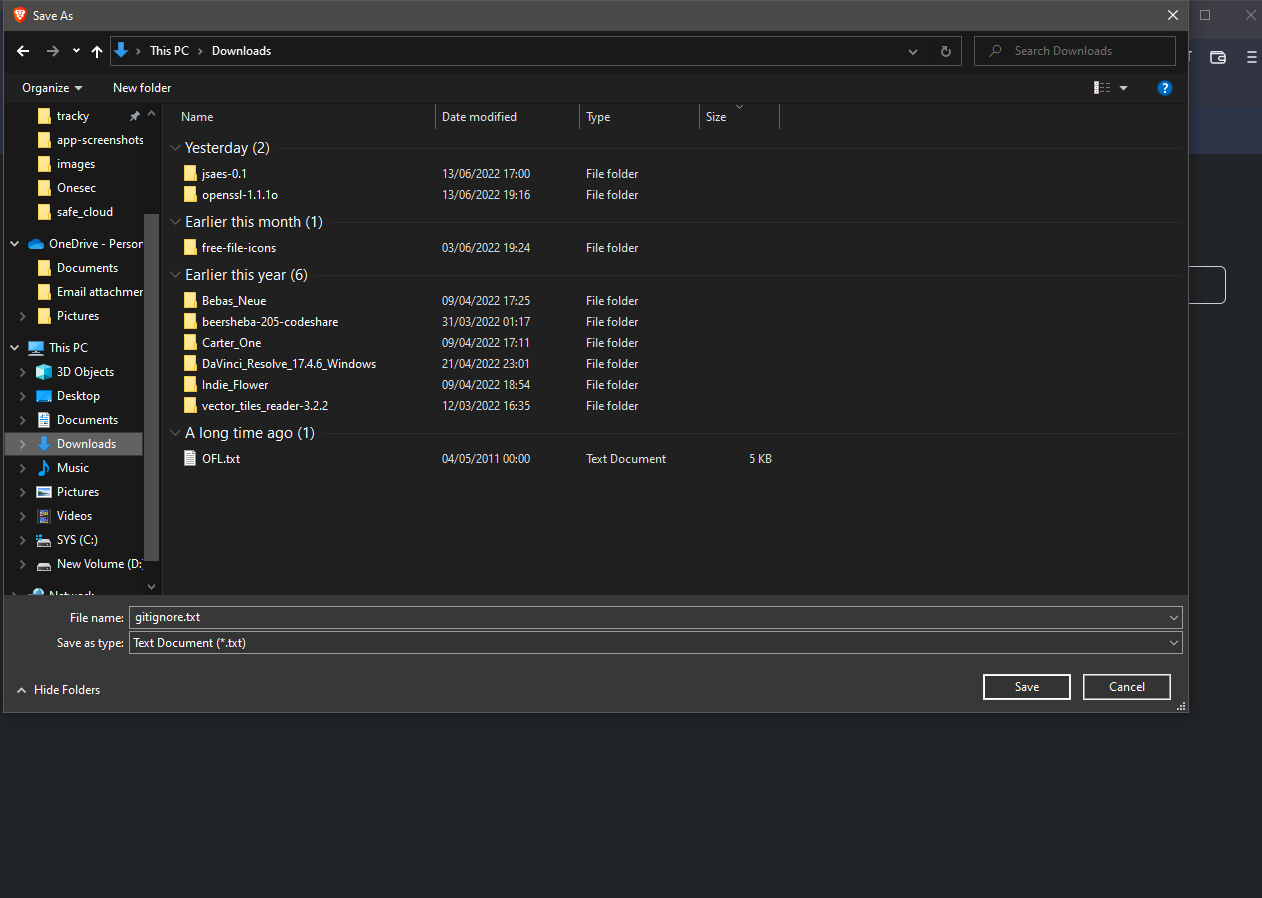


נכנסתי למשתמש שכבר יש בו קבצים, אז זה טען את הקבצים הקודמים. אם נרצה להוסיף קובץ, נגרור את הקובץ לתוך המסגרת האמצעית:

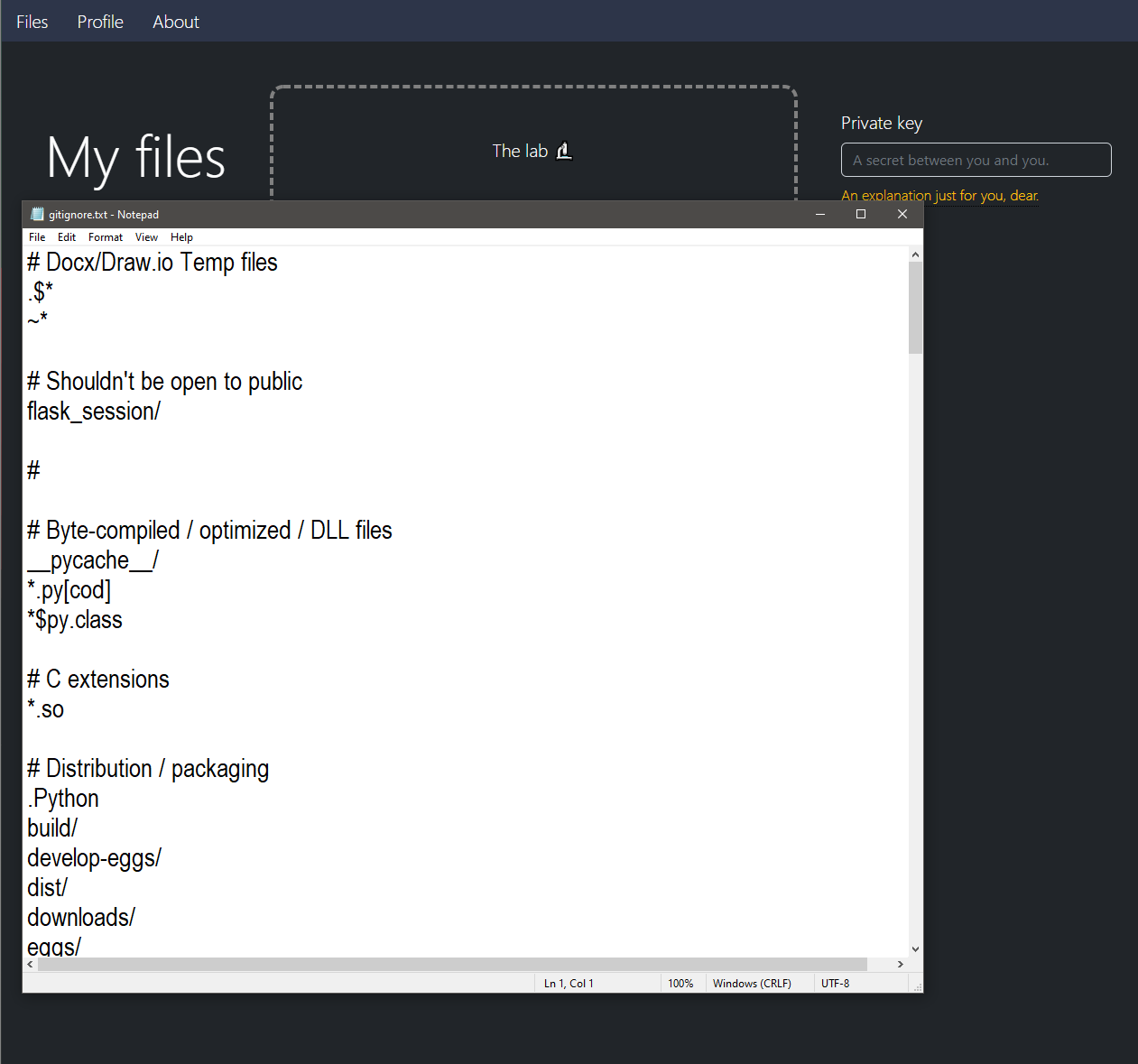


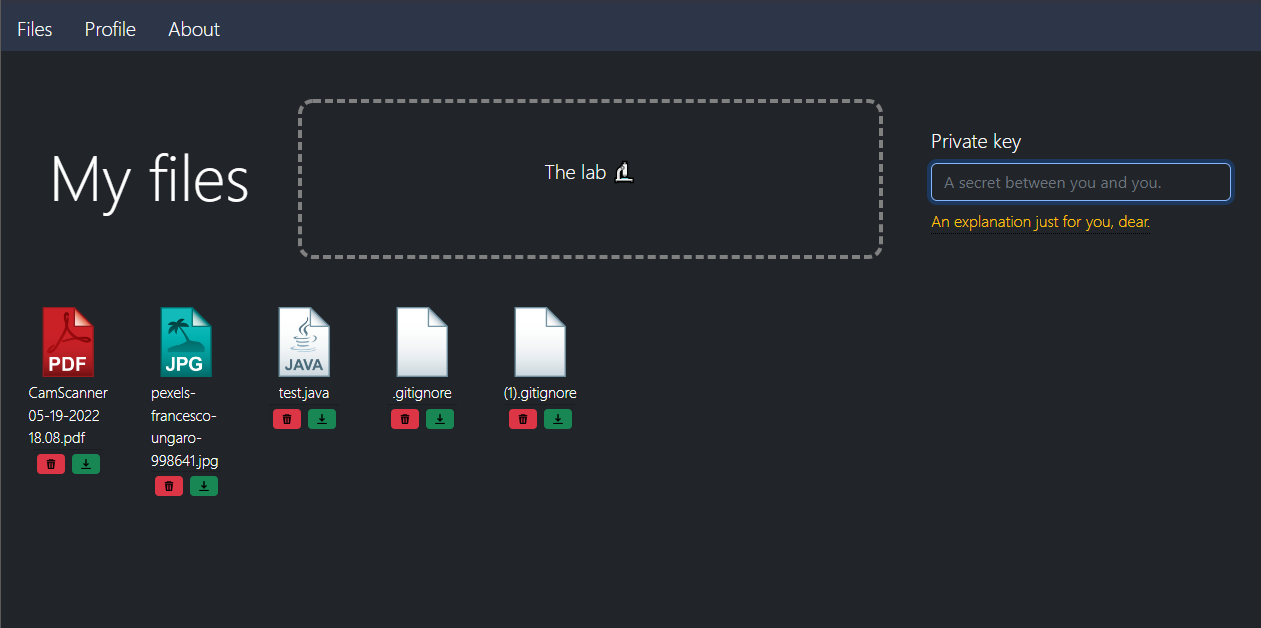
נראה שהקובץ אכן נוסף:



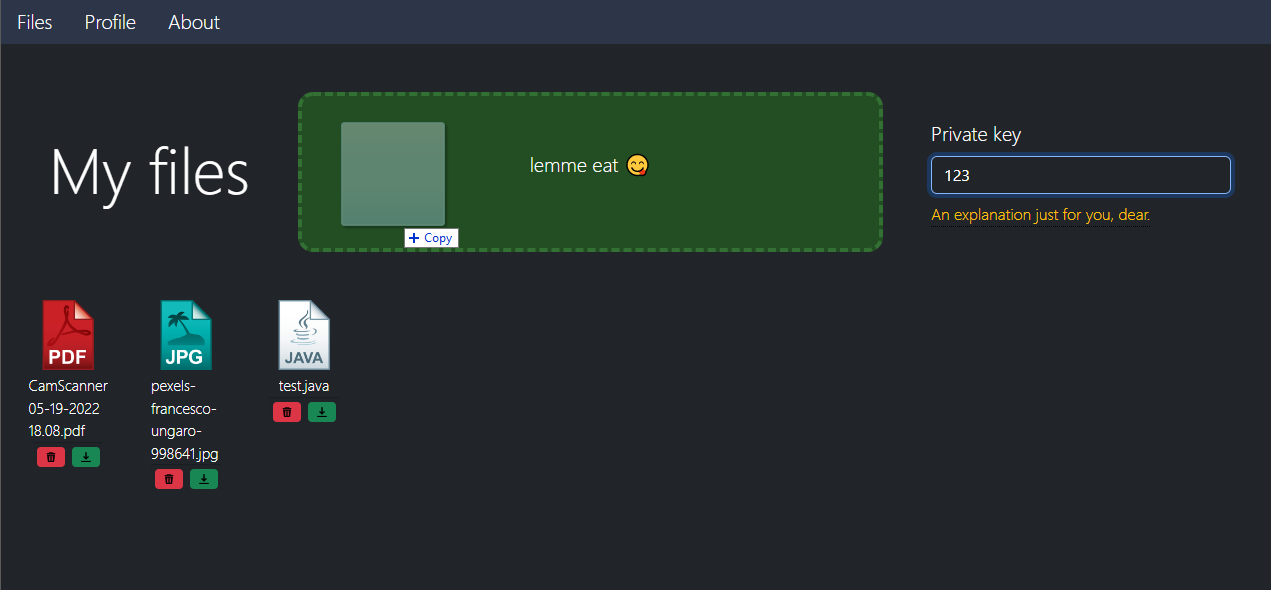
בנוסף, אם נרצה להוריד אותו, **נלחץ על הלחצן הירוק** ולמחיקה נלחץ על הלחצן האדום. בעת הורדה האתר יבקש ממנו את מיקום ההורדה: 

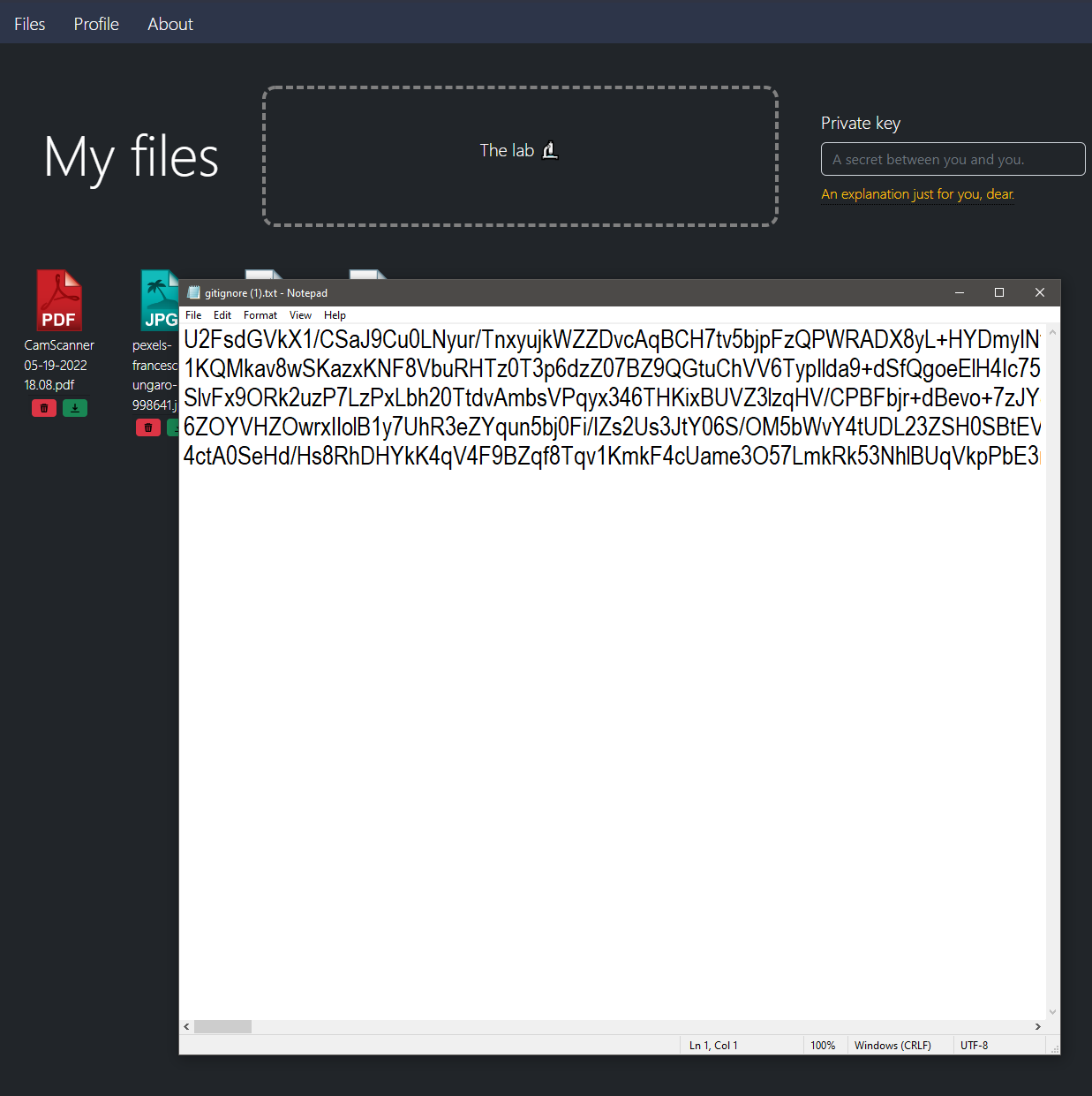
ונראה שהקובץ אכן הורד כמו שצריך.



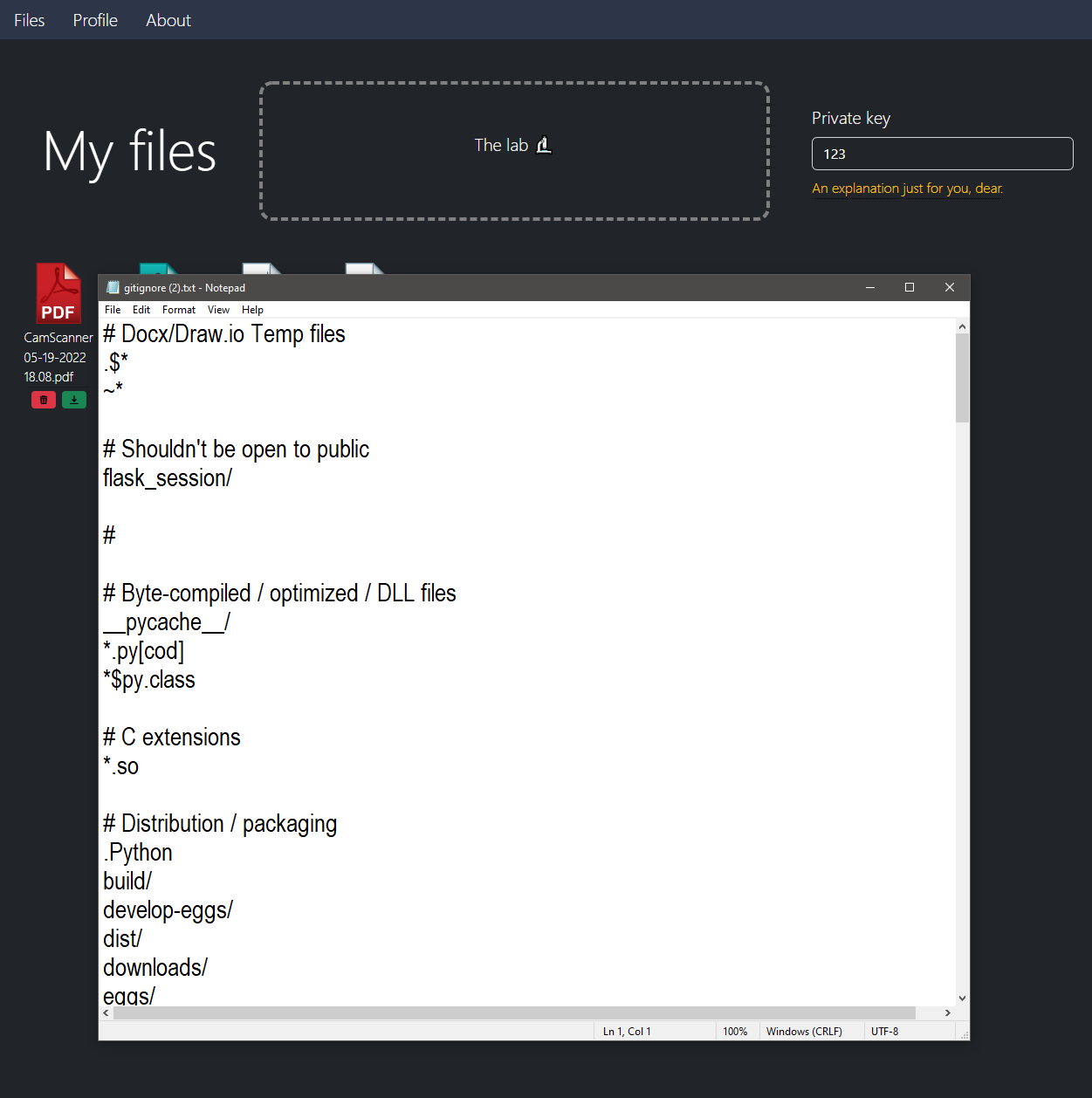
במקרה שנכניס את אותו קובץ פעמיים, נראה שהשם משתנה בהתאם: 

כמו כן, לדעתי הפיצ'ר הכי חזק בתוכנה הזאת, שאנחנו יכולים להצפין קובץ מצד הלקוח. כלומר, להצפין את הקובץ עוד לפני שיוצא מהמחשב שלנו – וכך אף אחד מלבדנו לא יוכל לדעת מה יש בקובץ, אלא אם כן הוא **מחזיק בקוד שלנו**. יש לזכור קוד הצפנה ולהוסיף קובץ כשהוא מוקלד:



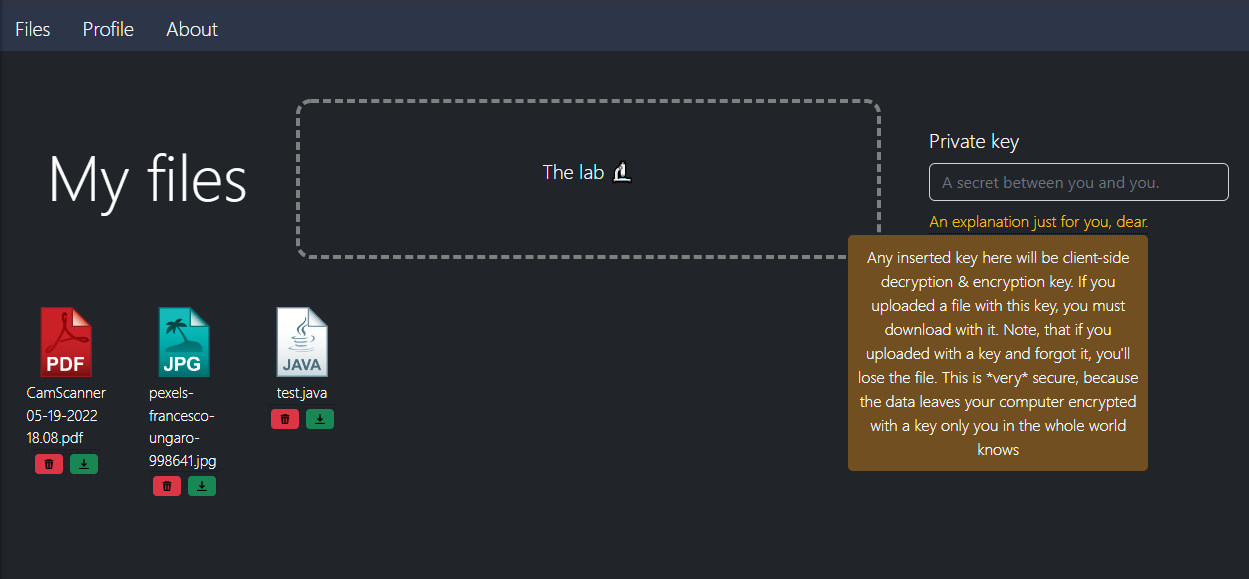
כעת, הקובץ שלנו נשלח בהצפנת AES עם המפתח "123". אם ננסה להוריד בלי המפתח: 

לא נצליח לקרוא את הקובץ. אך אם נוריד את הקובץ עם המפתח:

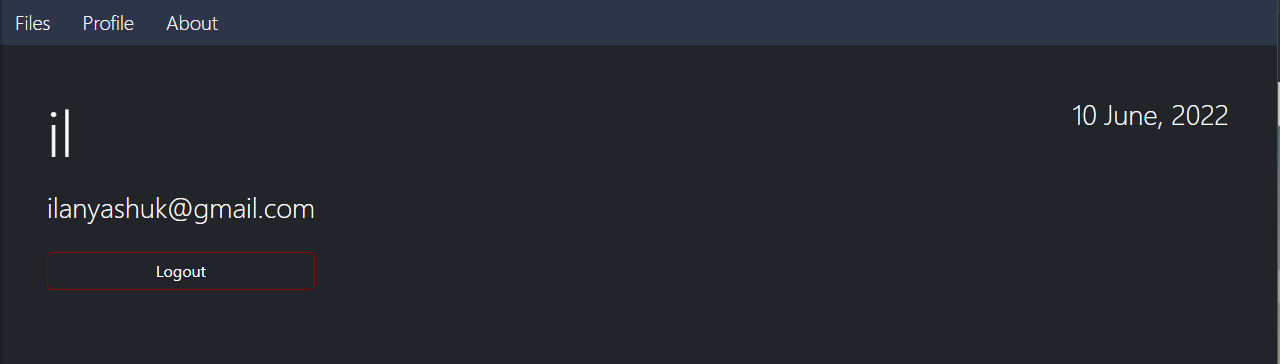


בום!

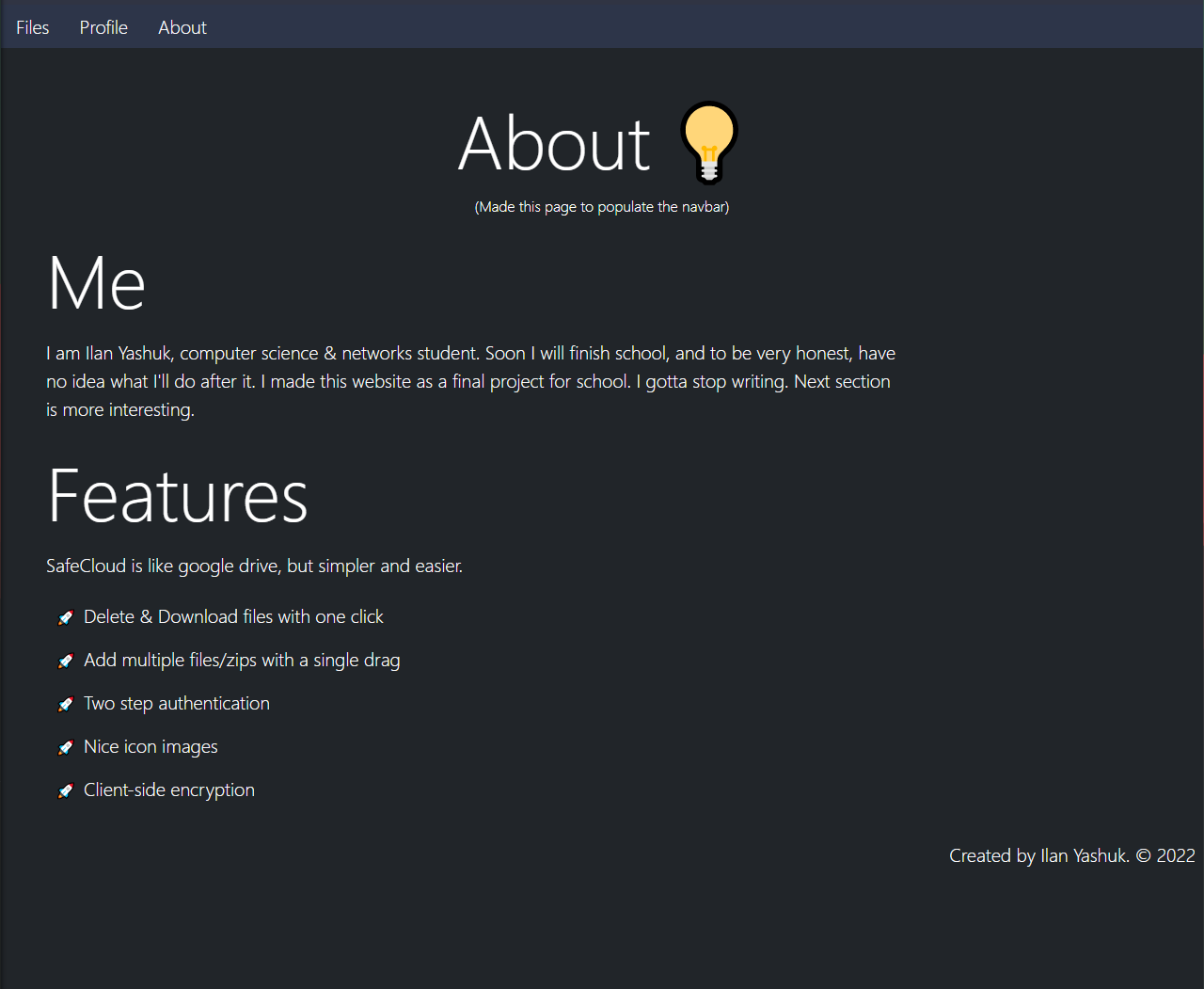
חוץ מזה, אציין שעוד פיצ'ר נחמד הוא שאפשר לגרור כמה קבצים ביחד (וזה עובד גם עם הצפנה). אם המשתמש יצטרך הסבר יותר מפורט, הוא יוכל להצביע על הטקסט הכתום:



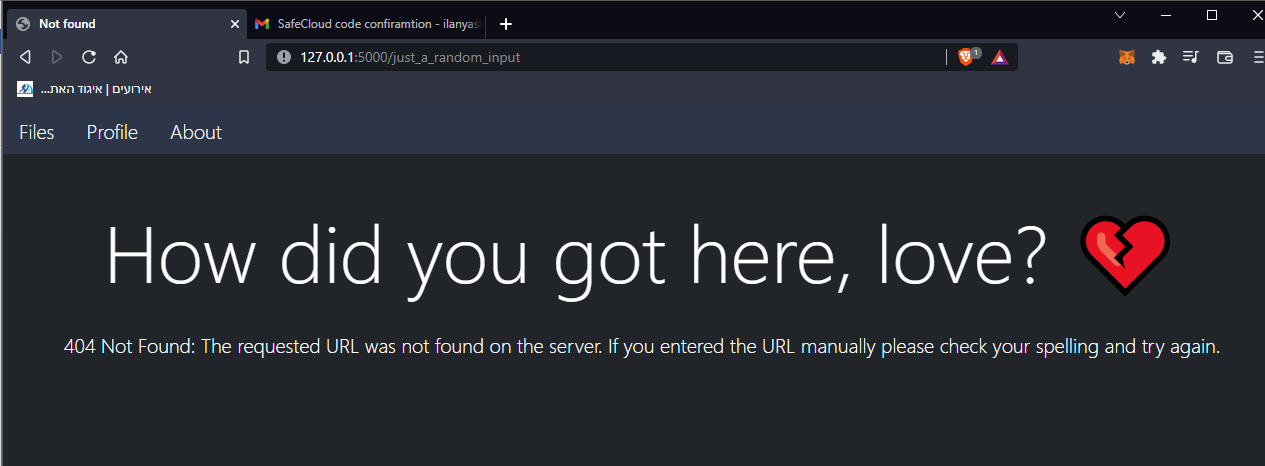
עוד עמוד מעניין הוא **עמוד הפרופיל**, שלא מכיל הרבה אבל מכיל את האפשרות להתנתק מהמשתמש. בלחיצה על הכפתור, השרת ישכח את המשתמש ואת ה "session" עמו, עד להתחברות חדשה.



שני הדפים האחרונים הם **דף שמסביר על פונקציונליות האתר:**



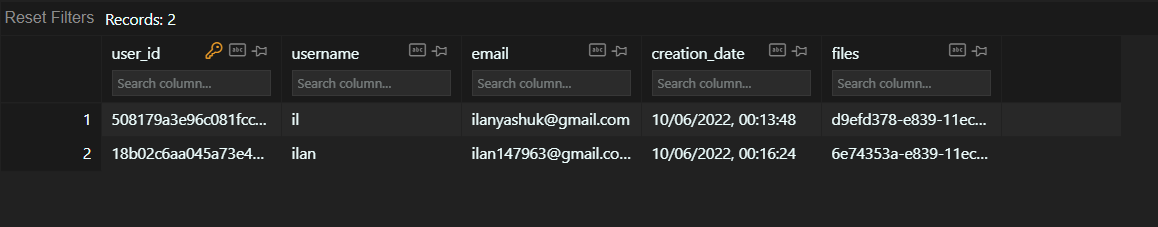
**ודף שגיאה**:



**בסיס הנתונים – sqlite3**

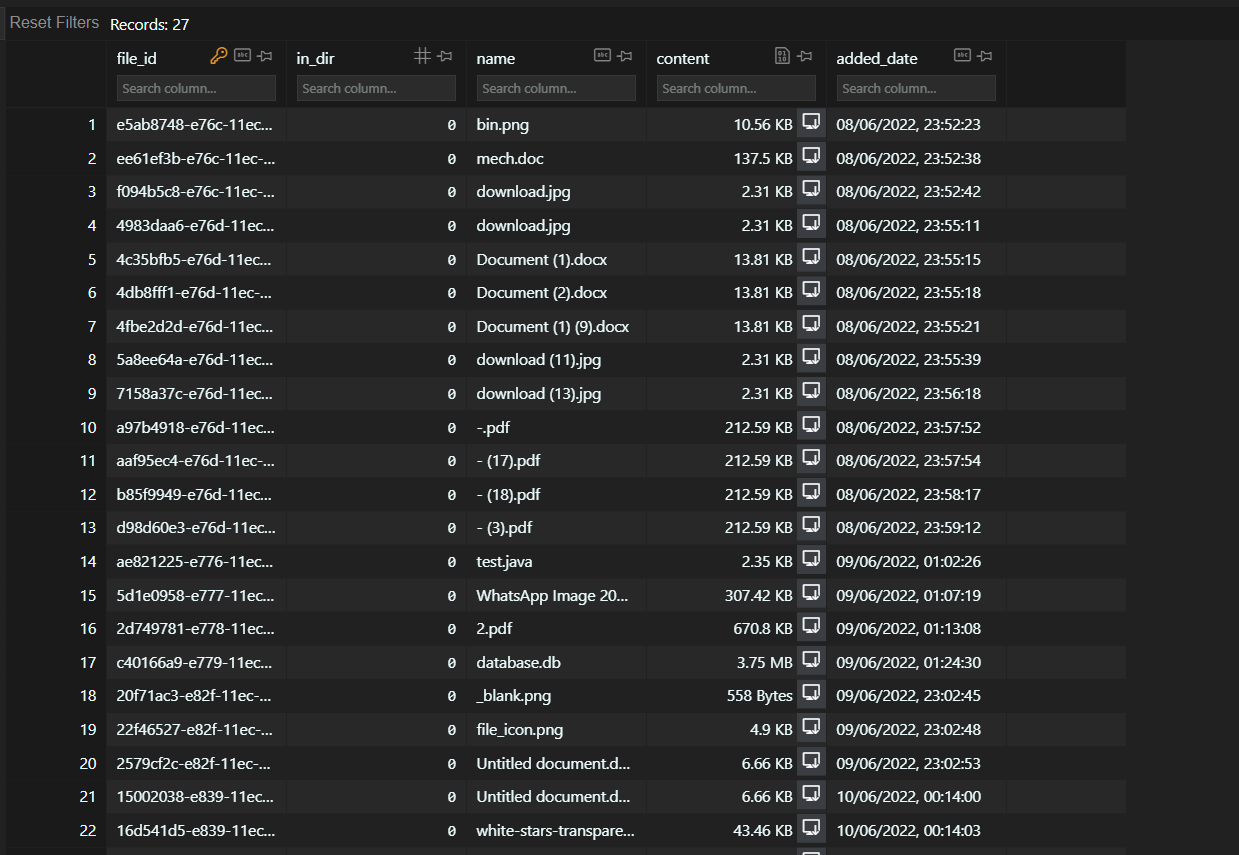
בסך הכל יש רק שתי טבלאות במאגר הנתונים. טבלת משתמשים וטבלת קבצים:

**טבלת משתמשים (דוגמה)**



* מס' מזהה למשתמש, מיוצר ע"י גיבוב סיסמה, תאריך יצירה, ואימייל, מפתח ראשי
* שם משתמש, לא מוצפן
* אימייל, גם לא יכול להיות מוצפן בשביל אימות דו-שלבי
* תאריך יצירה
* קבצים, מחרוזת המכילה את כל המס' הייחודיים של הקבצים של אותו משתמש

**טבלת קבצים (דוגמה)**



* מס' ייחודי, מבוסס זמן ולכן ייחודי לכל קובץ, מפתח ראשי
* האם בתוך תיקייה, למקרה של מימוש תיקיות בעתיד
* שם הקובץ
* תוכן הקובץ (כאן נכנסת ההצפנה אם המשתמש יבחר)
* תאריך הוספה

**מדריך למפתח**

תחת כותרת זו אסביר ואראה את המוח שמאחורי הפרויקט (כמובן שאני מדבר על המחשב). אני אראה קטעי קוד, קבצים שונים ואסביר עליהם את הרקע – שאדם שירצה אולי להמשיך ולפתח את התוכנה, יוכל לעשות זאת עם ההסברים פה.

**קובץ מס' 1 – DBCommands.py**

מהשם ניתן להבין שהוא אחראי על כל האינטגרציה עם מבנה הנתונים. הקוד שמטפל במאגר הנתונים משתמש במנעול אחד לכל המאגר, שלא ייגשו אליו שני משתמשים בו זמנית, ויפגעו במידע. לא אפרט בקובץ זה אך [מנעול](https://he.wikipedia.org/wiki/%D7%9E%D7%A0%D7%A2%D7%95%D7%9C_(%D7%AA%D7%95%D7%9B%D7%A0%D7%94)) משומש בשביל לשמור על משאב משותף לכמה תהליכים שרצים בו זמנית. לדוגמה, אם משתמש א' ומשתמש ב' יוספו למערכת בקירוב באותו הזמן, המערכת לא תדע את מי מהם לשמור. לכן, אחד מהמשתמשים ינעל מנעול על מאגר הנתונים, ולאחר שיסיים להשתמש בו – ישחרר את המנעול וייתן למשתמש אחר לגשת.

**קובץ מס' 2 – DBObjects.py**

בד"כ כשיוצרים מאגר נתונים, יוצרים אובייקטים שמטרתם לייצג את הטבלאות במאגר. כלומר אני יצרתי אובייקט למשתמש ואובייקט לקובץ, בשביל שאוכל לעבוד עם הקבצים שאני מכניס למאגר, בצורה תכנותית.

**רפלקציה**

**ביבליוגרפיה**

**נספחים**