gitguthub.
Creator: Dan Lvov

**פרויקט סוף שנה יא' 2021 – github**

**מגיש:**  דן לבוב

**תאריך**: 2021\5\31

**מורה:** ערן בינט

**שפה**Python 3.x

Contents

[**Project topic** 3](#_Toc73453866)

[Login / Register 3](#_Toc73453867)

[**Main page** 5](#_Toc73453868)

[Main page buttons 6](#_Toc73453869)

[“Show” buttons example 7](#_Toc73453870)

[Commit 9](#_Toc73453871)

[Branch off 9](#_Toc73453872)

[Merge 10](#_Toc73453873)

[Data structure and protocols 11](#_Toc73453874)

[**Logic and algorithms** 13](#_Toc73453875)

[File managing 13](#_Toc73453876)

[File history 13](#_Toc73453877)

[Commit delta 13](#_Toc73453878)

[Reload image (graph) 13](#_Toc73453879)

[List commit files 13](#_Toc73453880)

[What are archives good for and how do we use them? 14](#_Toc73453881)

[Why do we not use hashing? 14](#_Toc73453882)

[Who owns what? 14](#_Toc73453883)

[**Database Structure** 15](#_Toc73453884)

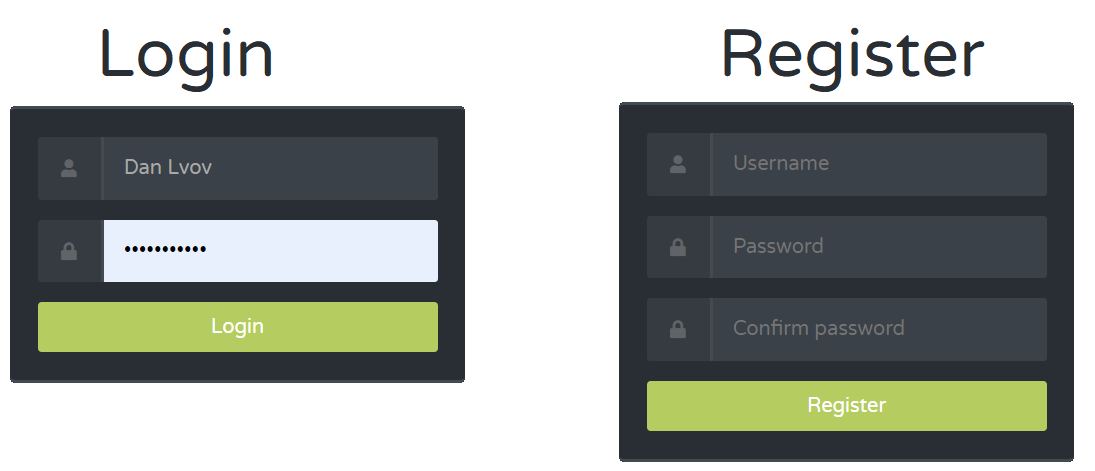
[Project dependencies 16](#_Toc73453885)

# **Project topic**

נושא העבודה הוא GITHUB. אתר המאפשר ניהול וורסיות (Version Control), שבו אפשר ליצור רפו (repo), ואליו להעלות קבצים ותקיות. ברפו יכולים להיות וורסיות שונות (branch). אפשר לשלב בין גרסאות, וליצור גרסאות חדשות בשביל שמספר אנשים יעבדו בו-זמנית ללא להפריע אחד לשני.

## Login / Register

כשנכנסים לאתר, אם קיים כבר משתמש הוא יועבר לעמוד שלו (הפרופיל שלו, שבוא אפשר לגשת לפרויקטים בהם הוא משתתף). אם לא קיים משתמש, יועבר לעמוד הLOGIN. אם המשתמש עוד לא נרשם, יכול לעבור לעמוד ההרשמה בלחיצת הכפתור "Not yet registered?".

****

Diagram

Description automatically generatedבפרויקט זה היה דגש על שימוש בSQL. SQL היא שפה המשמשת בתכנות ומיועדת לניהול נתונים האגורים בrelational database.

# **Main page**

Upon logging in, the user can view their profile. In the navigation menu, there are hyperlinks for viewing the other pages of the website. The two Admin-only pages are Browse Users and Admin View. " Browse users" displays the entire database as a JSON object. Admin View allows the admin user to view the database data and visualize every project.

Graphical user interface

Description automatically generated

The user has a drop-down menu to render any project they can access. If the user does not own or have access to any repo, a text will replace the drop-down menu, stating that they need to create a repo. After rendering a repo, Lisa from the Simpsons will present it.

Graphical user interface

Description automatically generated with medium confidence

## Main page buttons

Graphical user interface, application

Description automatically generatedAfter selecting a commit by clicking on it, the user can access the buttons and information on the right side.

At the top, the name, branch, owner, and comment of the commit are displayed.

Pressing “Show Changes” allows the user to view the difference between the previous commit and the current one.

Pressing “Show Files” displays the list of files in the current commit, with buttons to download and see the change history.

After the info box, there are three functions at the user’s disposal.

Commit, Branch, and Merge.

Graphical user interface, application

Description automatically generated

### “Show” buttons example

Graphical user interface, application

Description automatically generated

### Commit

**Graphical user interface, application

Description automatically generated**

Pressing on the commit button, the user sees a pop-up, allowing them to create a new commit. The user must enter a commit name and select the ZIP file containing the commit files. A description is optional.

### Branch off

Graphical user interface, application

Description automatically generated

This button lets the user create a new branch, copying the commit it branched off from.

### Merge

**Graphical user interface, text, application

Description automatically generated**Upon clicking the merge button, the user is presented with the following pop-up:

The merge popup is comprised of 3 parts. On the left side, a list of the current commit’s files. On the right, a button to view the commit tree. After clicking it, it will expand to reveal the repo visualized, with the selected commit highlighted.

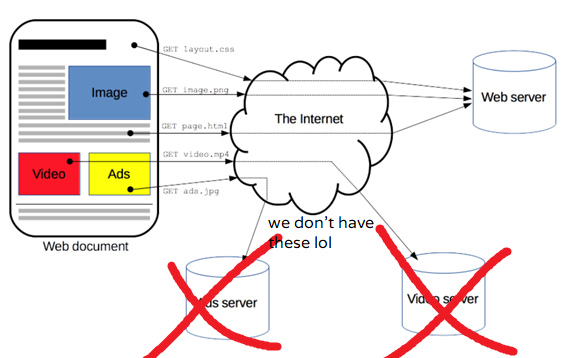
Graphical user interface, application

Description automatically generated

Below it are displayed the changes that will occur if the user confirms the merge action.

# Data structure and protocols

Since the project runs on HTTP (managed by Flask) it gets images, html files, and style(css) files by the GET method, and in input fields (E.g., Logging in: username, password) through POST.



Flask handles requests through function decorators, and internal features.

Text

Description automatically generated

These features include but are not limited to: “session”, “flash”, and all of flask’s functions for handling URLS.

**Session** is a dictionary which can be accessed in the HTML template that flask loads. This template may include snippets of code in flasks’ unique syntax.

**Flash** is a function which allows the programmer to store messages that can be displayed on the user’s next loading of the page. These messages have multiple categories.

Examples of URL handling features would be **redirect**, and **url\_for**. Error handling is also included:

Graphical user interface, text

Description automatically generated

Other than the database, there is also a class/object-oriented data structure which is synced to the database. Every class has its own helper functions to help manipulate the database. A repo class has an array of branch classes, which have an array of commit classes. Each with their respective properties.

Diagram

Description automatically generated

# **Logic and algorithms**

## File managing

To save space, we store all commits in a .zip archive. When scanning a commit’s files to calculate commit delta (diff) or for other purposes, the program opens a temporary folder in /temp\_zip. That folder will have the format: purpose\_id\_timestamp. This way, the folder name will never be the same, and scans will not overlap.

### File history

Every time a commit is selected, a function finds the chain of commits starting from the commit the user has selected, back to the first commit of the repo (origin point). Then, a scan is run for every file, to see if it has changed in the previous commits. Then, the function returns a list of the file history for every file scanned.

### Commit delta

Both commits are opened in a temporary folder and are scanned by a function that returns the files as a dictionary of directories, with each directory holding an array of file descriptors and their names. Then, a simple for-loop is run to see which files are missing, added, or have changed.

### Reload image (graph)

To make sure that the http will not send an “unchanged” code on the image, thus not updating it – we must rename the file. To do this, we find the current user’s image (the username is included in the filename), copy the data, delete the file, and rewrite it with the current timestamp. This way, the HTTP protocol will send a new image that the browser is forced to load anew.

### List commit files

To list the commit files (without mentioning any paths or directories) we simply walk in the directory using os.walk() and return a list with all the file names.

### What are archives good for and how do we use them?

As mentioned at the beginning of the **File Managing** chapter, this project stores user files in .zip archives to save space; This introduces the shutil module. The project includes a function used to make life easier with the shutil.py module. Instead of 4 vaguely named parameters, it has 2: a source folder and a destination.

The other use for saving files in archives is for ease of download. Having a version of the commit in archive, single-file form allows the user to download a commit in its entirety with a single click.

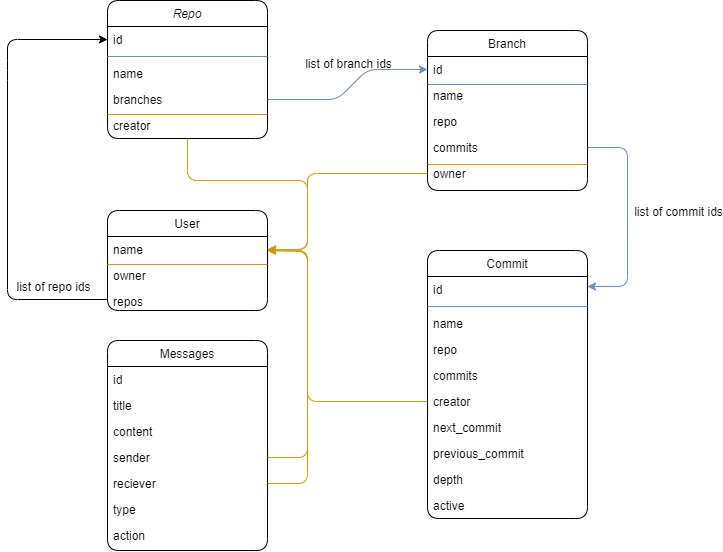
### Why do we not use hashing?

Because I am lazy, and this is a school project.

## Who owns what?

Every repo, branch, and commit have and owner value. When selecting a branch that the user does not own, a warning message shows up that every action will be made as a suggestion. This suggestion is sent to the branch owner as a message in the mailbox. The user can accept the message or ignore it.

# **Database Structure**



An empty version of the database is included in the submitted files. To interact with it, download DB Browser.

## Project dependencies

json

re

time

os.path -> join

os

shutil

cv2

numpy

json -> load

sqlite3

json