

# Design Doc for p00 by DECK, pd. 5

↳ Roster: Cindy Liu, David Lee, Ethan Cheung, Kalimul Kaif

## **PROJECT NAME:** "tragedy??"

A Flask application utilizing SQLite databases to display stored contents of wiki pages pertaining to tragic accidents, where readers can create or log into accounts to contribute to existing knowledge. (<https://www.emdat.be> could work for data)

**TARGET SHIP DATE:** {2025-11-14}

---

## Program Components:

- sqlite3 (backend data storage system)
  - user\_profiles (to store login information)
  - wiki\_pages (to store information for each page)
  - contributions (to store list of all contributions, ever)
- Python (application layer)
  - app.py (runs everything else)
- Flask (web server/delivery framework)
- html (frontend display)
  - homepage.html (starting page)
  - login.html (checks entered login info against database)
  - register.html (adds new login info to database)
  - profile.html (displays user history)
  - edit.html (allows user to make changes to text)
  - site.html (displays text)

**Our MVP:** A functional wiki with user login functionalities (including viewable profiles) and a functional logging system to track contributions to each wiki page

We are using three technologies: SQLite, Python (to execute scripts to serve HTML web pages), and Flask (to serve dynamic HTML pages).

We want our backend data storage system (using SQLite) to store multiple types of information that make up our wiki and its functionalities, including but not limited to (for now): **user\_profiles**, which is essentially a master list of all users and related information (PASSWORDS, profile biographies, date of

creation); **wiki\_pages**, which is a master list of all pages of the wiki with related information (CONTENT, last contributor, last edit date, compounded edit history); and **contributions**, which is a master list of all contributions ever made (timestamped, linked to a wiki page, linked to a user).

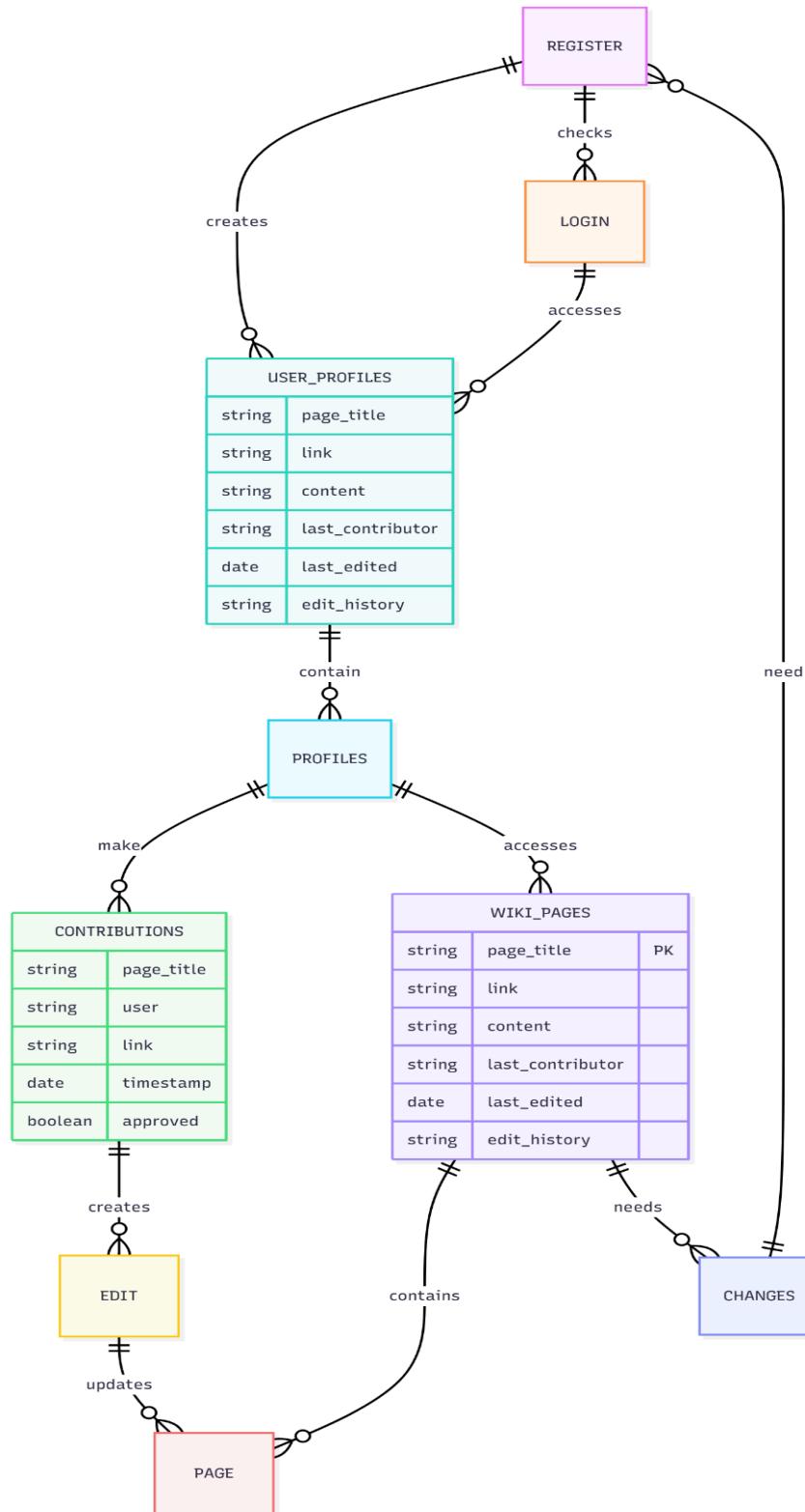
**user\_profiles** is utilized by Flask though a Python script for the pages *login.html* (to retrieve and compare user input login to system stored login info, maybe reset password if needed), *register.html* (to see if user already exists, append new information), *profile.html* (display contents of **user\_profiles** in profile html page), and *edit.html* (make sure user is logged in to edit, etc.).

**wiki\_pages** is used for *homepage.html* (list all the wiki pages available for viewing), *edit.html* (a specific entry of **wiki\_pages** is being edited, so we must retrieve the data), and *site.html* (displays an entry of the **wiki\_pages** database with contents, etc.).

**contributions** is used for *profile.html* (loop through all contributions, look for matching username to append to a list of that user's contributions), *edit.html* (append new contributions after *edit.html* form is submitted), and *site.html* (retrieve most recent contribution for that wiki page)

All pages are dynamic in nature, because they each depend on at least one database for the addition or removal of displayed contents on each page.

## Component Map :



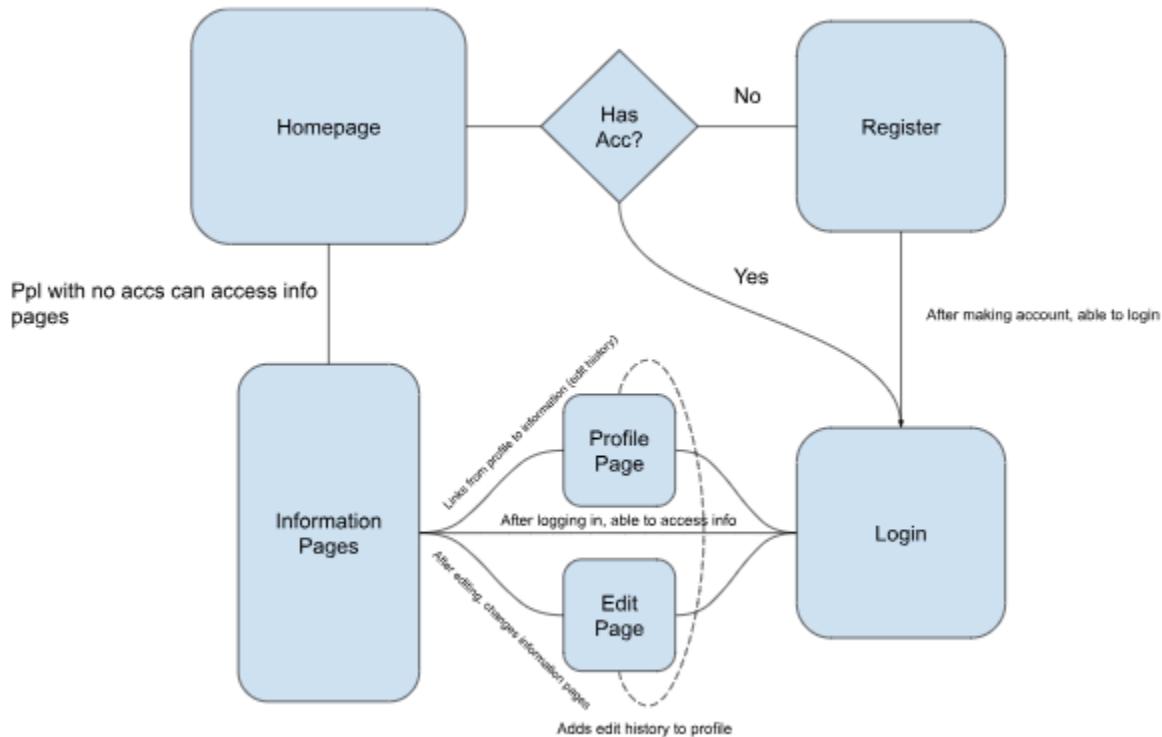
## Database Organization:

<b>USER_PROFILES</b>			
string	user	PK	
string	email		2fa? secondary priority
string	password		
string	bio		for customization of profiles
date	creation_date		only populated once during acc. creation
boolean	admin?		concept: admins approve contributions; secondary priority

<b>WIKI_PAGES</b>			
string	page_title	PK	
string	link		
string	content		string, assuming images can be added as links for variety of content on each page
string	last_contributor	FK	updates every edit
date	last_edited		updates every edit
string	edit_history		string that is appended to after every edit (mark contributor + last_edited and then update); secondary priority

<b>CONTRIBUTIONS</b>			
string	page_title	FK	
string	user	FK	loop through user entries to find contributions made by a single user
string	link		
date	timestamp		
boolean	approved?		

## Site Map:



## Breakdown of Tasks:

- create `__init__.py` (ethan)
- create csv files (david)
- setting up sqlite3 databases in `app.py` (david)

creating templates + corresponding functions in `app.py`:

- `homepage.html` (cindy)
- `login.html` (kalimul)
- `register.html` (kalimul)
- `profile.html` (david)
- `edit.html` (ethan)
- `site.html` (cindy)
- A lot of testing (communal..)