

# P01: ArRESTed Development

## Design Doc by MKED, pd. 5

↳ Roster: Kalimul Kaif, Mottaqi Abedin, Ethan Saldanha, David Lee

**PROJECT NAME:** "LiveMKED"

**DESCRIPTION:** A web application that helps users find ideal places to live in the USA by combining job opportunities, salary data and weather conditions. Users can search for specific jobs and filter locations based on multiple lifestyle and financial factors.

**TARGET SHIP DATE:** 2025-12-22

---

## Program Components

- sqlite3** (backend data storage system)
  - users** (stores user accounts, preferences, and saved searches)
  - saved\_locations** (stores locations bookmarked by users)
  - search\_history** (stores past searches for quick access)
- Python** (application layer)
  - \_\_init\_\_.py** (runs Python)
- Flask** (web server/delivery framework)
- External APIs**
  - Google Maps API** (location data, regional information)
  - OpenWeather API** (weather patterns, climate data for locations)
  - Indeed API** (job postings, location-based job availability)
  - PayScale API** (salary ranges for jobs by location to perform comparisons)
- html** (frontend display)
  - login.html** (checks entered login info against database)
  - register.html** (adds new login info to database)
  - homepage.html** (main dashboard with search/filter feature)
  - profile.html** (user profile with bio & preferences (job, salary, region))
  - edit\_profile.html** (update user profile data)
  - search\_results.html** (displays filtered job locations with all relevant data)
  - saved\_results.html** (displays bookmarked locations in chronological order)

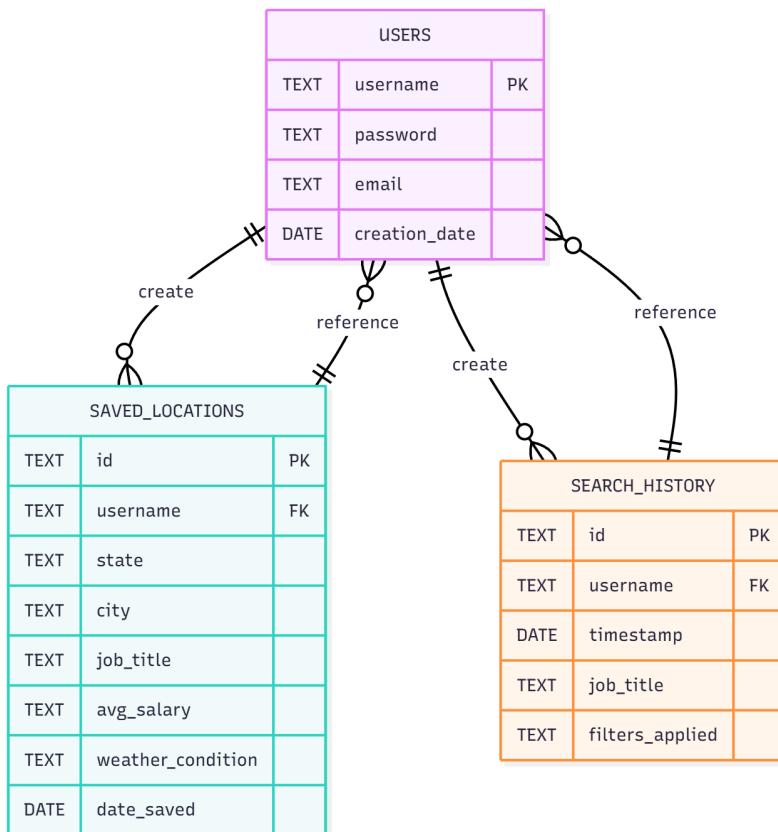
## Our MVP:

A platform to search for locations to settle in across the United States:

- Search for job postings and relevant details for a specific occupation
  - eg. hours per week, required skills, job benefits
- View and compare salary ranges for those jobs in different locations
- Option to filter results by state and weather conditions
- Save interesting locations for later review
- Compare multiple locations side-by-side

## Component Map

Disclaimer: We do not necessarily need more than three database tables to store all relevant information, because the most important components of each API call will be extracted and stored concisely in saved\_locations and search\_history to reduce redundancy.



# Database Organization

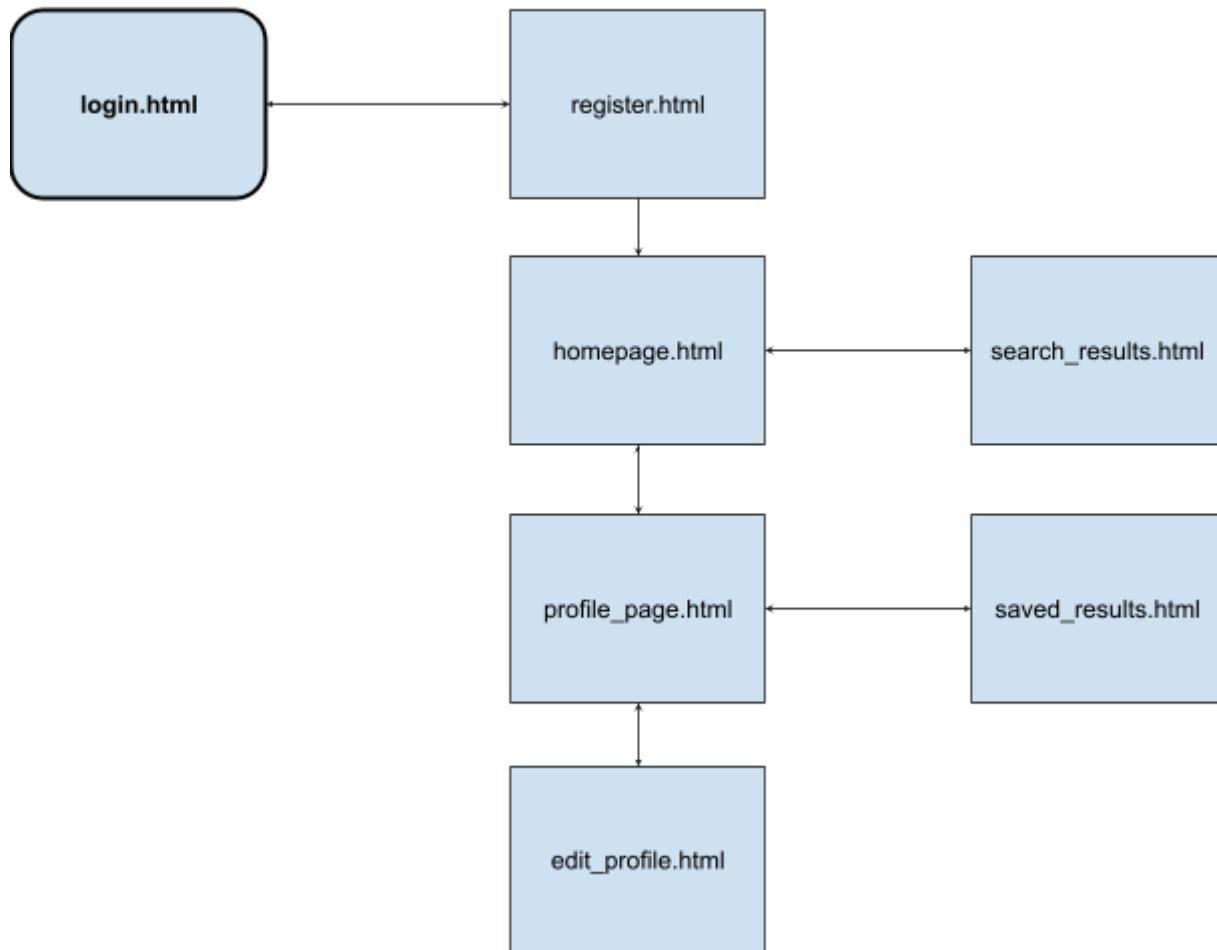
| <b>users</b> a master list of all registered users |               |    |   |
|--|---------------|----|---|
| TEXT   | username      | PK | unique identifier of blog user              |
| TEXT   | password      |    |   |
| TEXT   | email         |    |   |
| DATE   | creation_date |    | only populated once during account creation |

| <b>saved_locations</b> stores locations of interest bookmarked by users |                   |    |                                     |
|---|-------------------|----|-------------------------------------|
| TEXT  | id                | PK | unique identifier of saved_location |
| TEXT  | username          | FK | ties this saved_location to a user  |
| TEXT  | state             |    |                                     |
| TEXT  | city              |    |                                     |
| TEXT  | job_title         |    |                                     |
| TEXT  | avg_salary        |    |                                     |
| TEXT  | weather_condition |    |                                     |
| DATE  | date_saved        |    |                                     |

| <b>search_history</b> stores what searches the user has made in our website |                 |    |   |
|---|-----------------|----|---|
| TEXT  | id              | PK | unique identifier of search                           |
| TEXT  | username        | FK | ties this search to a user                            |
| DATE  | timestamp       |    | used for maintaining chronology of searches by a user |
| TEXT  | job_title       |    | what job did they search for?                         |
| TEXT  | filters_applied |    | all applied filters in a text format                  |

# Site Map

Everything EXCEPT login.html or register.html can route back to login via the logout feature



## Tasks & Assignments

- install guides & launch codes (David)
- requirements.txt
- setting up sqlite3 databases in app.py (David)

creating templates + corresponding functions in [app.py](#):

- login.html (kalimul)
- register.html (kalimul)
- homepage.html (david, mottaqi)
- profile\_page.html (ethan)

- edit\_profile.html (ethan)
- search\_results.html (david, mottaqi)
- saved\_results.html (kalimul, ethan)

## APIs

1. User submits search from the homepage (search contains job title + any filters)
2. API Calls
  - a. Indeed API (job postings by location)
  - b. PayScale API (salaries for occupation by location)
  - c. OpenWeather API (climate data for each location)
  - d. Google Maps API (location data, pictures of neighborhoods)
3. Store results in search\_history and, if saved by the user, in saved\_locations
4. Display results on an HTML page with organized user interface

## Front End Framework

We'll be using TailwindCSS as our main front end framework to style our website. We prefer using TailwindCSS over other front-end frameworks because it is very consistent in its utility classes and formatting, so it is easily replicable for UI components from one to another (utility classes can carry over with a few short, parsed keywords).