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: 2025-12-19

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)

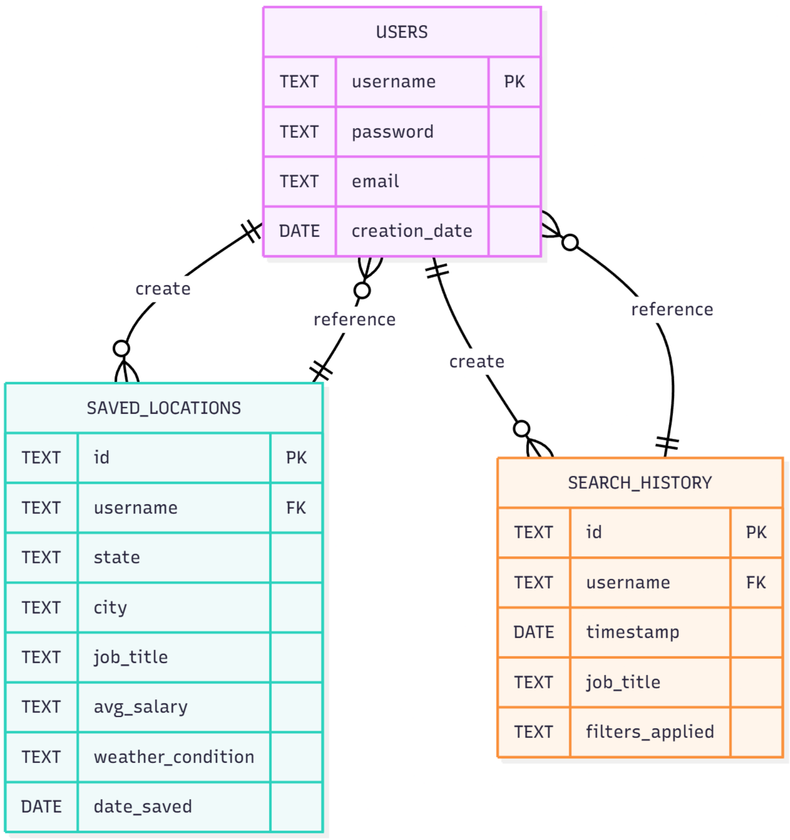
<u>Our MVP</u>:

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.



USERS		
TEXT	username	PK
TEXT	password	
TEXT	email	
DATE	creation_date	

SAVED_LOCATIONS		
TEXT	id	PK
TEXT	username	FK
TEXT	state	
TEXT	city	
TEXT	job_title	
TEXT	avg_salary	
TEXT	weather_condition	
DATE	date_saved	

SEARCH_HISTORY		
TEXT	id	PK
TEXT	username	FK
DATE	timestamp	
TEXT	job_title	
TEXT	filters_applied	

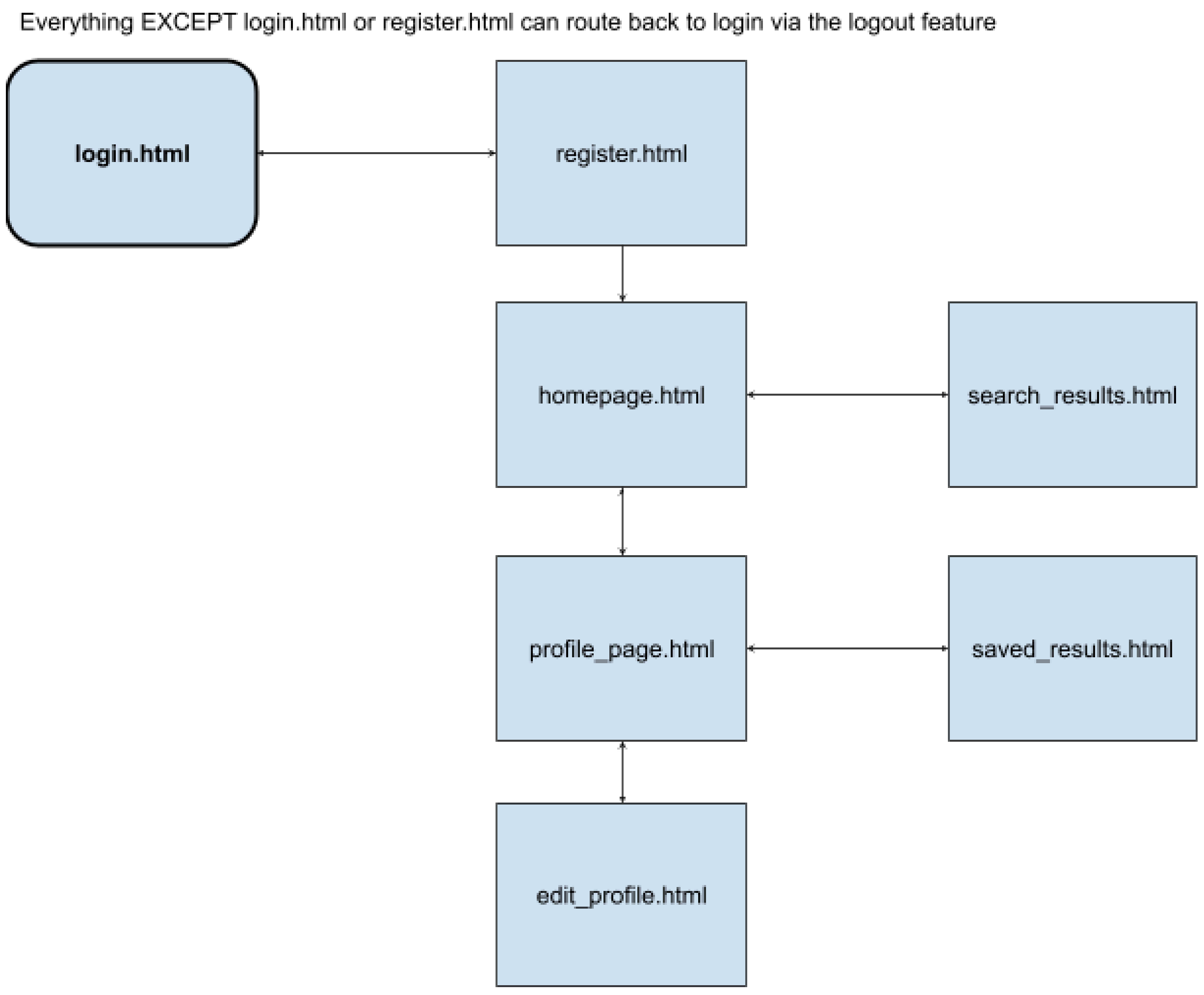
Database Organization

users 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

saved_locations 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		

search_history 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



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).