**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 a job based on keywords & location in the US. Users can search for specific jobs and filter postings based on various factors and criteria.

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

---

# Program Components

- ☐ sqlite3 (backend data storage system)
  - ☐ users (user account info)
  - ☐ saved_jobs (locations bookmarked by users)
  - ☐ search_history (past searches for quick access)
  - ☐ job_views (user statistics i.e. total jobs viewed, saved, and applied to)
- ☐ Python (application layer)
  - ☐ __init__.py (runs Python)
- ☐ Flask (web server/delivery framework)
- ☐ External APIs
  - ☐ Overpass API (location data, regional information)
  - ☐ JoinRise API (job postings, location-based job availability)
  - ☐ USAJOBS API (USA FED jobs)
    - Position Title
    - Apply Link
    - Location (lat, long)
    - Organization name
    - Position schedule (full time/part time, etc.)
    - Qualification summary
    - Position start and end date
    - Position Requirements [pull the entirety of details]
- ☐ html (frontend display)
  - base.html
  - register.html (adds new login info to database)
  - login.html (checks entered login info against 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_.html (displays filtered job locations with all relevant data)
  - saved_jobs.html (bookmarked locations in chronological order)
  - job_detail.html (displays job data from API call and location on a map embed)

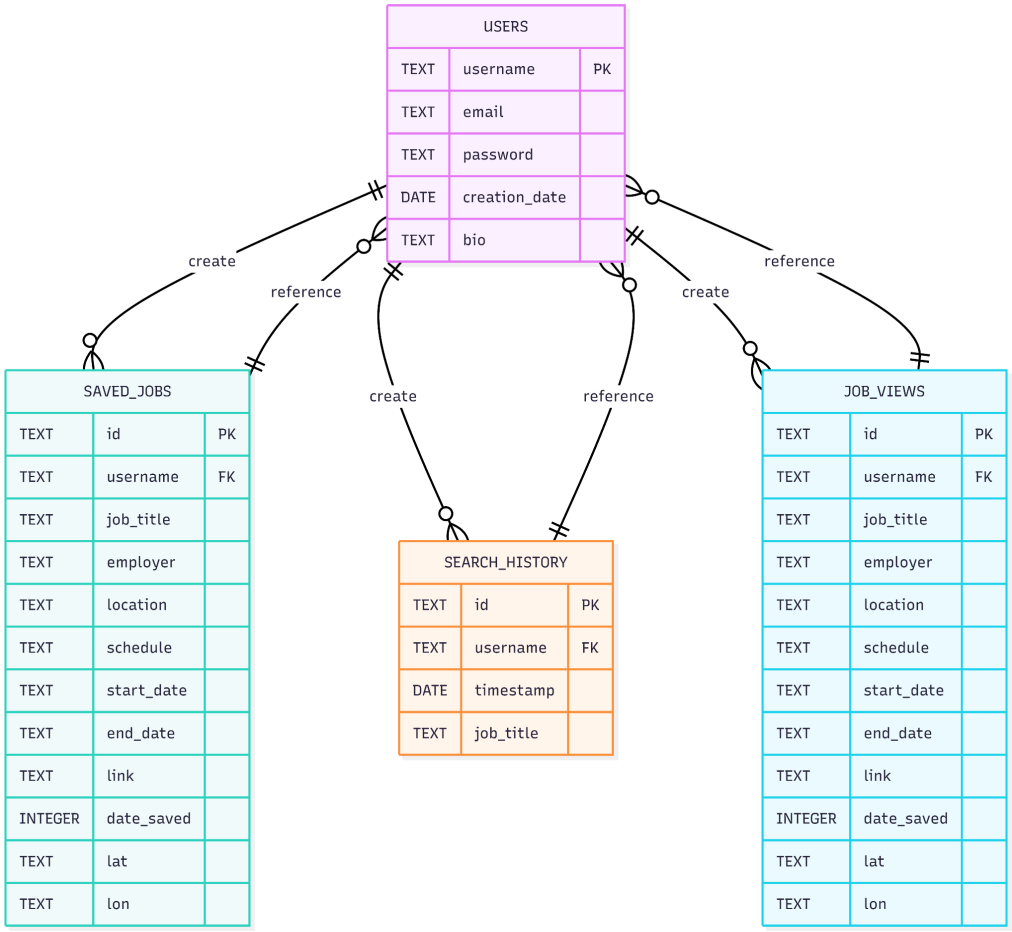
**Our MVP**:

A platform to search for job postings across the United States:

- Search for job postings for a specific occupation in a specific location in the US
- View and compare salary ranges for different postings side-by-side
- Save interesting jobs for later review and mark jobs that have already been applied to

# Component Map

Disclaimer: We do not necessarily need more than four 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

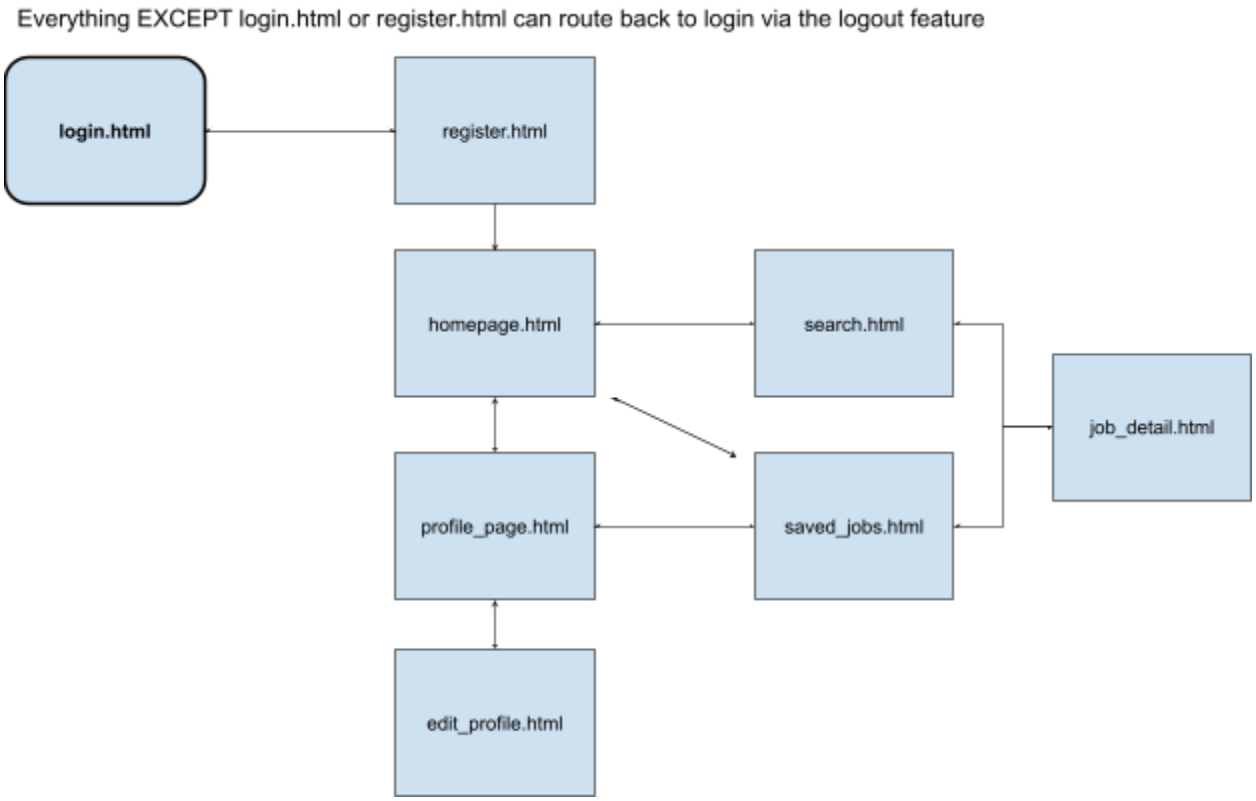
| **users** a master list of all registered users | | | |
|---|---|---|---|
| TEXT | username | PK | unique identifier of blog user |
| TEXT | email | | |
| TEXT | password | | |
| DATE | creation_date | | only populated once during account creation |
| TEXT | bio | | |

| **saved_jobs** stores jobs of interest bookmarked by users | | | |
|---|---|---|---|
| TEXT | id | PK | unique identifier for saved job |
| TEXT | username | FK | ties this saved_job to a user |
| TEXT | job_title | | |
| TEXT | employer | | |
| TEXT | location | | |
| TEXT | schedule | | |
| TEXT | start_date | | |
| TEXT | end_date | | |
| TEXT | link | | |
| INTEGER | date_saved | | |
| TEXT | lat | | |
| TEXT | lon | | |

| **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? |

| **job_views** all viewings of jobs assigned to each user, for “Recently Viewed” functionality | | | |
|---|---|---|---|
| TEXT | id | PK | unique identifier for saved job |
| TEXT | username | FK | ties this saved_job to a user |
| TEXT | job_title | | |
| TEXT | employer | | |
| TEXT | location | | |
| TEXT | schedule | | |
| TEXT | start_date | | |
| TEXT | end_date | | |
| TEXT | link | | |
| INTEGER | timestamp | | |
| TEXT | lat | | |
| TEXT | lon | | |

# Site Map




# APIs

1. User submits search from the homepage (search contains job title + any filters)
2. API Calls
    a. Overpass API (location data, fetched using a given latitude and longitude)
    b. Rise API (job postings, including descriptions and skill requirements, by city)
    c. USAJobs API (dataset on salary ranges for specific jobs in different locations)
3. Store results in search_history and, if saved by the user, in saved_jobs
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).

# 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)