

Immigo

1. Purpose & Motivation

Immigo is a platform designed to make it easier for immigrants to access information about immigration. It provides users with information across legal resources including recent immigration-related bills, organizations for legal aid, and community events. Target users are immigrants seeking reliable information to help with their transition. This project is important because it offers a centralized place to access multiple sources that empower immigrants with the support they need.

2. User Stories

The design of this website is organized very succinctly. As someone interested in immigration, especially with the recent political climate, I think this website is informative. I like how easy the website aims to encourage personal action in educating ourselves on immigrant rights. For feedback, I would want a feature that allows me to perform a simple search or filter my results so that I can access information easily.

As a user, I want to learn about local organizations that support immigrants so that I can connect with them for assistance. On the Organizations page, I hope to a list of groups with their logos, descriptions, and contact information. I want to easily identify which organizations offer the services I need. This helps me build a support network in my new community.

I think it would be nice to view a list of upcoming events in the events page so visitors can participate in community activities. The events page could have a grid of events with images and descriptions. Moreover, it would be nice to easily find dates, locations, and details for each event.

As a user of your website, I want the website to be very visually appealing and easy to navigate so that I can find information quickly and easily. The site should have a neat and clean layout and also still have a completely working design for mobile devices. I want to move between pages seamlessly, maybe through a navigation bar. This improves my overall experience and would encourage me to return to your site as a future user.

As a visitor, I want to understand the mission and purpose of this website so that I know how it can help me. I think it'd be nice for the home and about page to explain the goals of the site and its features, so I can make the most of the resources provided.

3. Architecture/Hosting Overview

- **Frontend:** React + Vite, Domain from Namecheap, AWS
- **Backend API Layer:** Flask, Unicorn, Docker, AWS EC2
- **Backend Database:** AWS RDS

4. API Documentation

Link: <https://documenter.getpostman.com/view/48953688/2sB3QGSAi5>(<https://documenter.getpostman.com/view/48953688/2sB3QGSAi5>)

- **Base URL:** `https://immigrow.site/`
- **Endpoints:**
 - `GET /orgs?page=1&per_page=15` → Get paginated organizations
 - `GET /orgs/{id}` → Fetch organization by ID
 - `GET /events?page=1&per_page=15` → Get paginated events
 - `GET /events/{id}` → Fetch event by ID
 - `GET /resources?page=1&per_page=15` → Get paginated resources
 - `GET /resources/{id}` → Fetch resources by ID

Pagination Response Format:

```
{
  "data": [...],
  "total": 100,
  "page": 1,
  "per_page": 15,
  "total_pages": 7
}
```

All APIs are hosted by Flask on an EC2 instance. Unit tests and Postman tests run in GitLab CI for each endpoint.

5. Data Scraping

ProPublica API - used to fetch nonprofit immigration related organizations Data Retrieved: Organization names, locations (city, state), EIN (tax ID), subsection codes (501c3), NTEE codes (topic classification), addresses, and links to IRS Form 990 tax filings and GuideStar profiles

CourtListener API v4 - used for fetching immigration related court cases and legal documents Data Retrieved: Case names, filing dates, court names, docket numbers,

judge names, legal citations, case opinions, and audio recordings of oral arguments

RAICES (Refugee and Immigrant Center for Education and Legal Services) Texas web scraper - used to scrape community events Data Retrieved: Event titles, dates, times, locations, descriptions, registration links, and event banner images Web scraped using BeautifulSoup

Mobilize America API - also used to gather immigration related community events Data Retrieved: Event titles, dates, start/end times, locations, venue names, descriptions, registration URLs, and featured images

REST APIS - ProPublica, CourtListener, Mobilize

6. Models and Instances

- **Events:** Represents nonprofit immigration-related organizations. Each instance corresponds to one organization, containing identifying information, mission classification, and reference materials. Instances usually include the organization's name, location including city and state, tax ID, IRS subsection type, NTEE topic classification, physical or mailing address, and external links such as IRS Form 990 filings or GuideStar profiles.
- **Resources:** Represents immigration-related legal resources from court data. Each instance corresponds to a single court case or legal document. Instances typically include the case name, court name, docket number, judge names, filing date, legal citations, and links to materials such as written opinions or audio recordings.
- **Organizations:** Represents community events relevant to immigration support and services. Each instance corresponds to a single event and includes information such as the event title, scheduled date, start time, physical location, event description, registration or sign-up links, and images if available.

7. Challenges

Some of our challenges included setting up backend hosting on AWS, identifying and linking instances with related attributes across multiple models, and integrating media content into the webpages. We overcame these challenges by consulting online resources such as AWS documentation, Geeks for Geeks help pages on HTML, using Gen AI to explain the process, and relying on each other's past knowledge.

There were a few challenges with the data sources. The EventBrite API could not be used to search for immigration related public events, so we had to switch the source. We decided on switching to web scraping from ImmigrationLawHelp.org but many events weren't formatted in the same structure making it hard to webscrape. Then we switched over to webscraping from RAICES, but there weren't enough relevant events (not in the 100s) so we paired that with the mobilize API to grab immigration related events. Also changed the source of organizations to ProPublica API due to conflicts with previous API decisions. Other challenges included actually retrieving two forms of media from our data sources, which we solved by delegating to different resources and APIs.

8. Phase II - Databases

Frontend: On the frontend we switched from hardcoded data to live data from our flash and APIs. We added a small API layer and now fetch all models showing loading spinners and error messages when things fail. Cards and detail pages render whatever fields the API returns. We also surface cross-links where available and show the total count of the resulted instances based on the API response. API Layer: The Flask API routes retrieve data from our various events, organizations, and resource tables in our RDS instance for the corresponding routes. This layer is hosted on EC2. Tables: Created Events, Resources, and Organizations table with 5 non null attributes. Each table also has 2 forms of media as features. There is a one to many relationship between Organizations and Events, a conjoint table between Events and Resources, and a conjoint table between Resources and Organizations to store the connections.

9. Phase II - Pagination

Backend Implementation: The backend API implements server-side pagination using SQLAlchemy's `.paginate()` method for efficient database queries. Each collection endpoint (`/api/orgs`, `/api/events`, `/api/resources`) accepts optional query parameters:

- `page` (default: 1) - Current page number
- `per_page` (default: 15) - Number of items per page

The API response format changed from a simple array to a structured object containing:

- `data` - Array of items for the current page
- `total` - Total number of items across all pages
- `page` - Current page number
- `per_page` - Items per page (always 15)
- `total_pages` - Total number of pages

This approach reduces payload size and improves performance by retrieving only the necessary data slice from the database.

Frontend Implementation: Created two reusable React components for pagination:

- `PaginationInfo` - Displays "Displaying X items" at the top of each model page, showing the actual count on the current page (e.g., "Displaying 3 events" on the last page)
- `Pagination` - Provides navigation controls at the bottom with First, Previous, Next, and Last buttons. Buttons are disabled appropriately at page boundaries (First/Previous on page 1, Next/Last on final page).

Each model page (Events, Resources, Organizations) integrates pagination with:

- State management tracking `currentPage`, `totalPages`, and `totalCount`
- Automatic data fetching when page changes via `useEffect` hook

- Display of total instance count in the page header
- Loading states and error handling for pagination requests

Data Distribution:

- Organizations: 57 instances across 4 pages (15, 15, 15, 12 items)
- Events: 48 instances across 4 pages (15, 15, 15, 3 items)
- Resources: 54 instances across 4 pages (15, 15, 15, 9 items)

Testing Coverage: Backend tests include 9 pytest tests covering both paginated and individual endpoints, verifying response structure, pagination metadata, and data accuracy. Postman collection includes 9 API tests with 3 dedicated pagination tests validating the response format. All existing frontend unit tests (10) and end-to-end Selenium tests (10) continue to pass without modification, as pagination is implemented transparently to the UI components.

10. Phase III - Search, Sort, and Filter

For Phase 3, we focused on making the site actually useful for finding specific information. The main challenge was implementing search functionality that works the way people expect - like Google - rather than just doing basic text matching.

Sorting and Filtering: Each model page now lets you sort and filter by at least 5 different attributes. We made sure to have a mix - can't just do all sorting or all filtering. For events, you can sort by date or title, and filter by location, timezone, and duration. Organizations can be sorted by name or city, filtered by state, topic, and size. Resources sort by publication date or title, and filter by topic, scope (Federal/State/Local), and court name.

The tricky part was getting the UI right. We didn't want filters that auto-reload on every click because that gets annoying fast. Instead, there's an "Apply Filters" button so you can select multiple things before triggering the search. The filters use checkboxes for multi-select (like picking multiple states) and dropdowns for single values. Backend-wise, we handle multiple filter values by accepting comma-separated lists in the query string.

Search Implementation: The search functionality was more involved than expected. Initially we just did basic SQL LIKE matching, but that doesn't work well when someone searches for "immigration clinic Texas" - they expect results containing all those words, even if not in that exact order.

We ended up implementing a relevance scoring system similar to how search engines work. When you search, the backend splits your query into individual words, then scores each result based on:

- Full phrase matches get 1000 points (highest priority)
- Each individual word match gets 100 points
- Consecutive word sequences get 50 bonus points

So if you search "legal clinic," a result with "free legal clinic" scores higher than one with "legal services" and "clinic hours" in different parts of the text. The backend sorts results by this score before sending them back.

On the frontend, we highlight matching text with yellow marks (like Ctrl+F in a browser). There's also a small indicator at the bottom of each card showing which fields matched your search - helpful when the match is in a field that's not displayed on the card, like a description or URL.

Global Search Page: The most significant feature is the new `/search` page accessible from the navbar. Instead of searching one model at a time, you can search everything at once. The page makes three API calls in parallel and displays results in three columns - Events, Organizations, and Resources.

Each column scrolls independently (max 800px before scrolling kicks in) and has its own pagination. So you might be on page 2 of Events while looking at page 1 of Organizations. We added vertical borders between columns because without them it looked like one big mess of cards.

The cards themselves took some iteration to get right. Initially they had different heights which looked terrible, so we set a minimum height of 260px. We also use flexbox to push content to the bottom of cards so there's no awkward whitespace in the middle. Organizations show a truncated description, resources show citation info - basically filling the space intelligently rather than leaving it empty.

Code Reusability: We built several reusable components during this phase:

- `SearchAndFilters.jsx` - The search/sort/filter UI used on all model pages
- `HighlightedText.jsx` - Wraps text and adds `<mark>` tags around matches
- `MatchIndicator.jsx` - Shows which fields matched (e.g., "Matches: Description, Title")
- `CheckboxFilter.jsx` - Multi-select checkbox component with scroll

These components are used identically on model pages, detail pages, and the global search page. Keeping the code DRY was important because we knew we'd be tweaking styling and it would be a pain to update in multiple places.

API Updates: Updated the API documentation in Postman with all the new query parameters. Each endpoint now accepts:

- `search` - Full-text search across all model fields
- `sort_by` - Field to sort by (e.g., "date", "name", "title")
- `sort_order` - "asc" or "desc"
- Model-specific filters (e.g., `location`, `state`, `topic`, `scope`)

The Postman collection includes tests for combined search + sort + filter operations to make sure everything works together.

Testing: Added 15 new backend tests specifically for Phase 3 functionality:

- 3 search tests (one per model)
- 3 sort tests (one per model)
- 3 filter tests (one per model)
- 3 combined operation tests
- 3 additional edge case tests

Frontend and Selenium tests from Phase 2 still pass. The Postman collection was updated with tests for the new query parameters.

Challenges: The relevance scoring algorithm took a few iterations. First attempt just counted word matches, but that ranked "the the the immigration" higher than "immigration clinic" which was wrong. Adding the phrase matching bonus fixed that. We also had to normalize whitespace because database text sometimes has weird spacing that broke exact phrase matching.

Getting the global search page layout right was harder than expected. Three equal columns looked good on desktop but terrible on mobile. Had to make it responsive so columns stack on smaller screens. The independent scrolling also caused some CSS headaches with pagination controls - we wanted them inside the scroll area

so they don't take up permanent space.

Performance-wise, making three parallel API calls is fast (thanks Promise.all), but we were worried about database load. The relevance scoring happens after retrieving results, not in SQL, which isn't ideal for huge datasets. For our current data size (50-60 instances per model) it's fine, but we'd need to optimize if the dataset grows significantly.

Overall, Phase 3 made the site way more functional. You can actually find specific information now instead of just browsing through pages.