ATX Assist
Group 10 Phase III Technical Report

Grace Choi, Eshi Kohli, Benjamin Zimmerman, Saketh Kotamraju, Tvisha Andharia

Website Link

http://www.atxassist.me/

Motivation

As Austin sees a growing influx of residents, an increasing number of low-income communities are facing displacement due to gentrification. This has caused the cost of living to soar, putting many individuals and families in difficult positions. Our project seeks to establish a dedicated space for underserved communities to learn about essential resources. By connecting Austin residents to information about affordable food, clothes, and housing, we hope to make an impact in our local community.

User Stories

Phase I:

Show a picture of an example of a thrift store on the home page

In response to this user story, we created a card for each of our models at the bottom of the home page. We feature an image of a thrift store on the thrift store card, which offers a sneak peek and serves as an engaging entry point for users.

Display a small slogan / title in big text on the home page

In response to this user story, we displayed a succinct slogan on the home page that encapsulates the mission and purpose of our platform. This addition serves to create a strong initial impact and communicates the essence of our services to users at a glance.

Show the number of total instances for each model at the top of their pages

We added the total number of instances available at the top of each model page. This provides users with a quick reference point, allowing them to gauge the scope of information within each category – be it food pantries, thrift stores, or affordable housing.

Add a brief description for each model page explaining the model

To enhance user understanding, we included concise descriptions for each model in the cards at the bottom of the home page. Additionally, we added an even shorter description at the top of each model page, offering users a clear overview of each respective model.

Show a picture of an example food pantry on the home page

In response to this user story, we created a card for each of our models at the bottom of the home page. We feature an image of a food pantry on the food pantry card, which offers a sneak peek and serves as an engaging entry point for users.

Phase II:

When showing the rating attribute on the model pages, could you display "(rating) / 5" instead of "rating."

In response to the user story regarding the rating attribute, we implemented a change to display the rating in the format "(rating) / 5" instead of just "rating." This modification increases clarity for users, providing a standardized representation of the rating scale across all model pages.

Could you add a light mode button for at least for the home page?

Following this user story, we incorporated a light mode button on the home page. Users can now toggle between light and dark modes, allowing for a personalized and visually comfortable browsing experience.

Could you make the description at the top of the about page left aligned instead of center aligned?

We adjusted the layout to align the description on the about page to the left. This change increases readability and creates a more conventional and user-friendly presentation.

Would it be possible to describe some attributes of the models on their respective pages? It could maybe go under the number of options displayed.

We added a short sentence describing the attributes of each model on their respective pages. As suggested, we put it under the text that shows the number of instances for each model.

For pantries and thrift stores, could you display the number of ratings as its own row in the information table for each instance page?

In response to this user story, we display the number of ratings as its own row in the information table for pantries and thrift stores. This addition provides a quick reference to the popularity and user feedback for each instance.

Phase III:

Fix Food Pantries Connections "Learn More"

In response to this user story, we ensured that all connections lead to relevant and informative pages. This fix enhances the user experience.

Get clearer quality images for home page models and background

Following this user, we upgraded the images to be of higher resolution and clarity. This enhancement enriches the website's aesthetic appeal and user engagement by offering a visually pleasing and professional appearance.

Have connections for thrift stores "Nearby resources"

In response to this user, we introduced a feature that displays links to resources in the vicinity of each thrift store. This addition facilitates user access to local services and information, enriching the browsing experience with practical and localized content.

Fix API for Thrift Stores page "404" error

To address this user story, we corrected the API endpoints. This rectification ensures a seamless and error-free navigation experience, reinforcing the website's reliability and user satisfaction.

Fix API Affordable Housing "404" error and connections "Nearby resources"

This user story was tackled by fixing the "404" errors encountered on the Affordable Housing page and by integrating "Nearby resources" connections. By resolving the API issues, we ensured uninterrupted access to affordable housing listings. Additionally, fixing the nearby resource connections provides users with valuable insights into local amenities and services, further supporting informed housing decisions.

Phase IV:

Add a short summary for each model at the top of each model

We added a one-line sentence that explains what users can expect to see for each model page. The sentence lives at the very top of every model page.

Create a custom mouse cursor for your website

We replaced the default cursor with a bluebonnet icon, which is a significant flower in Texas and can be found all around Austin. This ties into our focus on Austin communities.

For the Sort and Filter by option window, save the state of the window.

We pre-populated each search form with the value(s) from the user's last search.

For the Sort and Filter options window, please replace the "close" button with two new buttons, "apply" and "revert".

We replaced the "close" button with three: "Apply", which sends the search as configured; "Revert", which clears all filters; and "Close", which now actually closes the search window as implied (instead of its previous behavior which submitted the form when closed).

The atx assist main logo is clickable and yet does nothing. Please make it so that it navigates to one of the pages.

We updated the logo such that clicking on it takes you to the home page.

RESTful API

https://documenter.getpostman.com/view/32820631/2sA2r545aP

Our RESTful API has endpoints to GET all food pantries, thrift stores, and affordable housing instances. This API also has endpoints to GET specific instances of these models.

The API endpoints are as followed:

1. GET all food pantries
    a. https://api.atxassist.me/get-all-food-pantries/
    b. Returns a list of all food pantries stored in the database
2. GET food pantry
    a. https://api.atxassist.me/get-food-pantry-id/
    b. Returns a specific instance of food pantry, identified by its id
3. GET all affordable housing
    a. https://api.atxassist.me/get-all-affordable-housing/
    b. Returns a list of all affordable housing stored in the database
4. GET affordable housing
    a. https://api.atxassist.me/get-affordable-housing-id/
    b. Returns a specific instance of affordable housing, identified by its id
5. GET all thrift stores
    a. https://api.atxassist.me/get-all-thrifts/
    b. Returns a list of all thrift stores stored in the database
6. GET thrift store
    a. https://api.atxassist.me/get-thrift-id/
    b. Returns a specific instance of thrift store, identified by its id
7. GET search thrift stores
    a. https://api.atxassist.me/thrift-stores/search?query=[enter query]
    b. Returns instances of thrift stores that match the search query
8. GET search food pantries
    a. https://api.atxassist.me/food-pantries/search?query=[enter query]
    b. Returns instances of food pantries that match the search query
9. GET search affordable housing
    a. https://api.atxassist.me/affordable-housing/search?query=[enter query]
    b. Returns instances of affordable housing that match the search query
10. GET filter and sort thrift stores
    a. https://api.atxassist.me/thrift-stores/filtered
    b. Returns instances of thrift stores sorted or filtered by the specified query
11. GET filter and sort affordable housing
    a. https://api.atxassist.me/affordable-housing/filtered
    b. Returns instances of affordable housing sorted or filtered by the specified query

12. GET filter and sort food pantries
    a. https://api.atxassist.me/food-pantries/filtered
    b. Returns instances of food pantries sorted or filtered by the specified query

Models

Our project is utilizing three models: food pantries, affordable housing, and thrift stores. Each model has five or more attributes associated with it. The food pantry model will have the name of the food pantry, the location, the opening hours, a link to the website, the average rating, and the number of ratings given; the affordable housing model will have the name of the housing, the location, whether it's only for students, whether it is available to the disabled community, whether there is a waitlist, and a link to the website; the thrift store model will have the name of the thrift store, the location, the phone number, a link to the website, the average rating, and the number of ratings given.

Frontend Architecture

Our frontend is a React.js app, and we're hosting through AWS S3. Currently, we have a home page, an about page, and 3 model pages. Our 3 model pages (food pantries, affordable housing, and thrift stores) have several model instances. Each instance points to a new page with a unique ID to provide more information on the instance.

How it's running:

In order to run our app, clone the repository and run the following commands:

1. npm install
    a. This will install all the packages needed to run our application
2. npm start
    a. This will run the application on a development server in order to start making changes to the site and seeing those changes

Structure:

Our frontend structure is that of the following:

1. /src
    a. Holds all of our pages and tests
2. /src/assets
    a. Stores all of our images
3. /src/components
    a. Has all the subcomponents we use for our pages, including cards, model tiles, and the navbar

Testing

Unit tests of the TypeScript code are done using Jest, in a file named “App.test.tsx”. The file has eleven different tests, which each attempt to render a react component and determine if what is rendered matches what is expected by matching text in the document to an expected string. The tests ensure the rendering of the home page, navbar, about page, and model and instance pages given valid data. The tests can be executed by running “npm test”.

Acceptance tests are done through Selenium, originating in a file, “acceptance_tests.py”. This file uses a chrome driver to simulate our page, and tests are run to determine if the page has the elements that we are expecting. Currently, acceptance_tests.py tests navigating through different model pages from the navigation bar and the pagination of all the models. The tests can be run by the command “python3 acceptance_tests.py”.

All tests are incorporated into the test stage in the configuration YAML file that runs our pipeline in GitLab, ensuring continuous integration.

Sorting and Filtering

Our sorting and filtering features are located in FoodPantries.tsx, AffordableHousing.tsx, and ThriftStores.tsx. In each of these files, we have a dropdown with all the attributes we can sort or filter with (e.g. name, rating, etc.). When the button is clicked, we then do an API call.

Searching

Site searching is located in FoodPantries.tsx, AffordableHousing.tsx, ThriftStores.tsx, and NavBar.tsx. The search bar is in all model pages and in the navbar. The navbar search returns global results, while the model search returns model-specific results. The data is displayed in order of relevance with multi-word searches, giving first importance to the full phrase and then the amount of partial/individual words found.

Backend Architecture:

How it’s running:

The backend server is running on an EC2 instance listening on port 3003. AWS has a load balancer that redirects traffic to a target group containing one target, which is port 3003 on the above-mentioned EC2 instance.

Structure:

The Flask app itself can be found in /backend/server.py. Upon activating a virtual environment, we can install the Flask app as a package via ‘pip install’.

The two directories /backend/data and /backend/scrapers are scripts used in development to scrape third-party API’s for data. They should not be necessary to the continued operation of the project.

The structure of the database can be changed via add_to_tables.py.

Testing:

Unit tests of the RESTful API were exported from Postman into the file “ATX Assist API.postman_collection.json”. It contains a test for each endpoint of our RESTful API by checking if the GET request was successful and returned the expected data.

Our backend unit tests (done in test_api.py) were made to test each API endpoint function we created in our server.py flask file. We wanted to ensure each request acted as intended and have something for our pipeline to check to make sure new commits do not introduce unwanted bugs.

Sorting, Filtering, and Searching

We implemented features (sorting, filtering, and searching) to find all instances that contain a certain value. Sorting and filtering look through certain fields for said value, whereas searching looks through all fields. This was done in both SQL and Python, as these languages can lend themselves to different types of searches.

Database Architecture:

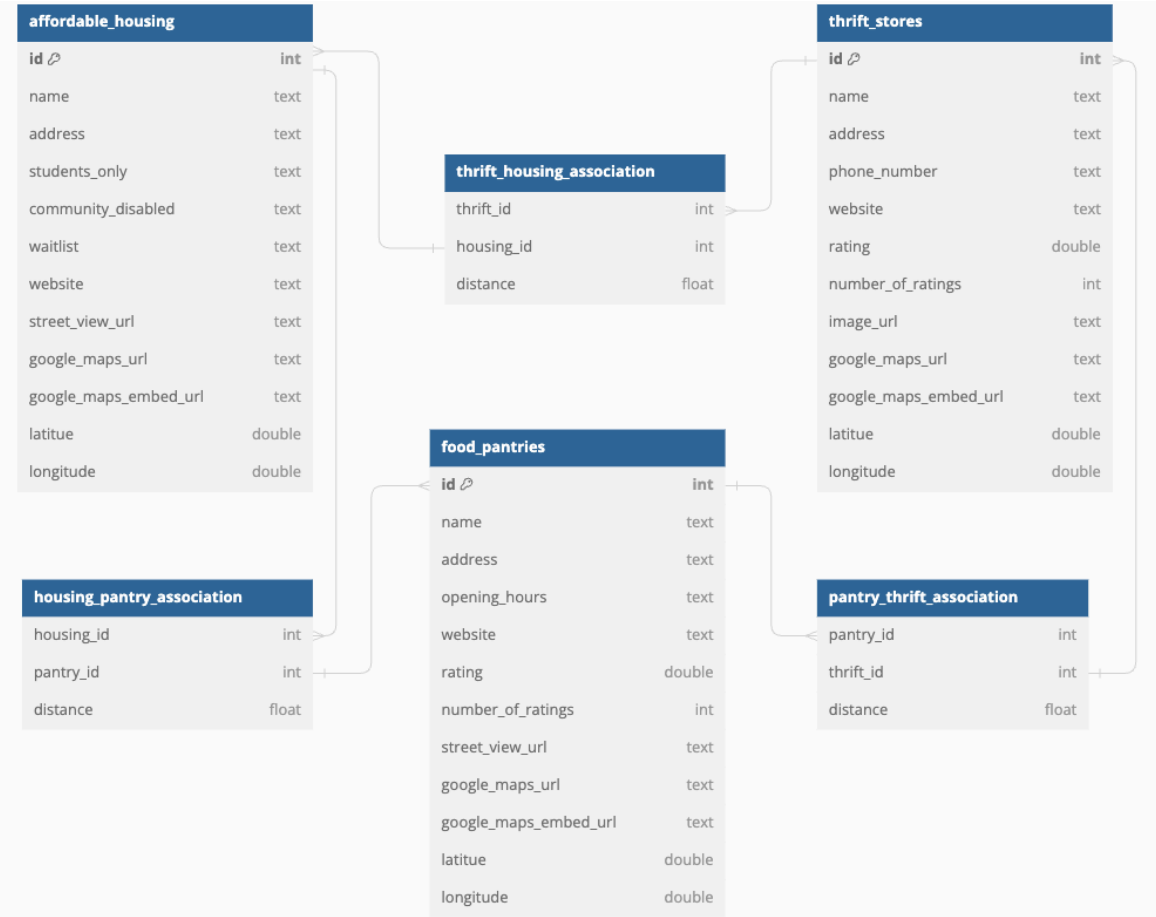
How it’s running:

Our database is hosted on AWS RDS and is connected to an AWS EC2 instance. The database has an endpoint on port 3306, the default port used for MySQL.

Structure:

In order to properly connect and push the data, we utilized SQL Workbench and SQLAlchemy. In our MySQL database, we have six tables (one for each model, and one for each combination of models): food_pantries, affordable_housing, thrift_stores, pantry_thrift_association, housing_pantry_association, thrift_housing_association.

The following UML diagram showcases the organization of each table in the database:



Tools

In this phase, we leveraged the following tools for various aspects of our project:

AWS API Gateway

- Enabled us to construct a publicly accessible REST/HTTPS API and seamlessly route it under our custom domain name.

AWS RDS

- Gave us the ability to store data in a MySQL relational database

GitLab

- Functioned as our repository for storing files related to the full-stack website and backend code. Additionally, it provided features for project planning through issue trackers and milestones.

Flask

- Implements the end point logic.

MySQL Workbench

- Supported database management.

SQLAlchemy

- Employed as an ORM to create a many-many relationship with models.

React

- Allowed us to create an interactive UI for our website with ease.

React-Bootstrap

- Provided a framework and design templates for enhancing the UI of our website.

React Router

- Facilitated efficient routing within our website.

Node js

- Enabled the development of the backend server responsible for processing the logic of our main API, supporting concurrent requests efficiently and ensuring seamless data loading for our website.

AWS Elastic Cloud Compute (EC2)

- Utilized for hosting our Node.js server and other backend code.

AWS Lambda Functions

- Implemented to connect with our API in AWS API Gateway. These functions routed API requests to the corresponding Node.js server endpoint and efficiently returned the required data.

Postman

- Utilized for documenting and testing our RESTful API.

AWS Simple Storage Service (S3)

- Hosted our frontend website.

Docker

- Utilized for containerization, for streamlined development, deployment, and scalability of our application across different environments.

D3.js

- JavaScript library that allows developers to create dynamic, interactive data visualizations in web browsers.

Zoom

- Employed for remote meetings to discuss various aspects of the project.

Slack

- Used as a communication platform for project-related discussions with the Teaching Assistants.

VS Code

- Served as an Integrated Development Environment (IDE) for debugging and version control using Git.

Ed Discussion

- Used to ask questions to peers and TAs.

Challenges

Throughout this phase, we encountered the following challenges:

Working with query strings:

- For requests with parameters, such as searches, sorts and filters, we used query strings to pass those parameters around as needed. We found that we needed to transform these query strings, both in form (string vs. map/dict) and in content (remapping some key-value pairs). To help with this, we used URL parsing and formatting libraries in both JavaScript and Python.

Unit tests:

- While adding frontend unit tests, we ran into issues with testing code that uses the JavaScript Fetch API, which is not defined within Jest execution. Although this seems to be a fairly common issue, we ran into issues with some of the most common solutions being out-of-date or incompatible with TypeScript. At first, we planned to use a library which relied on the order of API calls to return data. However, we ran into an issue with the model pages sending multiple API calls to different endpoints (to retrieve data for related instances) in an unpredictable order. Ultimately we decided on a slightly more custom approach which returns data based on the requested URI instead.

Site-wide dark mode customer user story:

- We struggled to complete this user story because of how we originally set up our dark mode feature for our Home page. We ended up requesting (and receiving) a new user story to work around this issue.

Limited items per request in Developer's RESTful API:

- We ran into the challenge with our developer's RESTful API having a limit of 10 items per request. This meant we had to make multiple requests to gather all the data we needed to make the visualizations for their site.