ParentAidATX Phase 2 Technical Report

Group Members: Amna Ali, Andrew Harvey, Rubi Rojas, Allison Still, Ethan Yu
Website Link: https://parentaidatx.me

CS373: Software Engineering
University of Texas at Austin

1. **Phase 2 Overview**
   a. This phase largely consisted of implementing the backend of our website. We created a Flask API hosted on AWS EC2 and a PostgreSQL database hosted on [Railway](#). We then consumed our API in the frontend to display and populate the models' cards and instance pages. Furthermore, we added pagination to these cards for a better user experience, as now there are over 300 instance pages and cards. Lastly, we added unit tests and acceptance tests for both the frontend and backend.
   b. **IMPORTANT CHANGE:** The original phase 1 Government Programs model was changed to Books about single parenting, housing, and childcare. Also, some model attributes were changed for refinement.

2. **Introduction**
   a. **Purpose**
      i. ParentAidATX empowers single parents in Austin, Texas, by simplifying access to essential resources, including family-related government assistance, affordable housing, and childcare services. Navigating support systems can be overwhelming, especially for single parents balancing work and family. ParentAidATX bridges this gap by connecting users to tailored programs and services, making it easier to understand eligibility, find nearby affordable housing, and locate quality childcare. We believe every parent deserves support, and every child deserves stability.

   b. **Questions Our Website Will Answer**
      i. What are some helpful books for single-parenting?
      ii. Where can I find affordable housing near my child's daycare or school?
      iii. What are the best childcare centers that fit my preferred times and style of education for my child?
      iv. What books are available for housing and childcare?

3. Models and Instances
  c. Model 1: Books
    i. **Instances:** ~100
    ii. **Attributes**
        1. Author
        2. Date of Publication
        3. Page Count
        4. Price
        5. Category (Parenting, Childcare, or Housing related)
        6. Description
        7. Link to Purchase Book
    iii. **Media**
        1. Links
        2. Text descriptions of the book
        3. Book Cover Image
    iv. **Connections to Other Models**
        1. Housing: Displays local programs based on location
        2. Affordable Childcare: Books related to childcare
  d. Model 2: Housing
    i. **Instances:** ~1000
    ii. **Attributes**
        1. Name
        2. Cost/Rate
        3. Rating
        4. Reviews
        5. Address
        6. Zip Code
        7. Style of Housing (Apartment, Condo, House)
        8. Crime Level
        9. Nearby Park
        10. Transportation
    iii. **Media**
        1. Google Maps for location
        2. Location Image
        3. Text description

iv. **Connections to Other Models**
        1. Books: Books programs pertaining to housing
        2. Affordable Childcare: Displays affordable housing near the location of schools/daycares

e. **Model 3: Childcare**
    i. **Instances:** ~8000
    ii. **Attributes**
        1. Name
        2. Age Range
        3. Open Time
        4. Close Time
        5. Program Type
        6. Image
        7. URL to daycare
        8. Address
    iii. **Media**
        1. Google Maps for location
        2. Image of Childcare Location
        3. Link to Childcare Website
    iv. **Connections to Other Models**
        1. Books: Books programs pertaining to childcare
        2. Housing: Affordable housing near the location of schools/daycare

4. User Stories

    f. "Make links for websites on model pages hyperlinks" - Sana Kohli

        i. We added a button to visit links from the model pages.

    g. "Show room options in each housing" - Sana Kohli

        i. This was not possible to implement in this phase, we will work with our user on refining this idea for the next phase.

    h. "Show at least 2 pictures per housing complex" - Sana Kohli

        i. We added multiple pictures.

    i. "Have the JSON responses from API return the IDs of the other model instances that are connected to this model instance" - Sana Kohli

        i. Added.

    j. "Show stars in the rating in the card page" - Sana Kohli

        i. This was replaced with the Open Time and Close Time for the childcare model. For the housing model, ratings are displayed.

5. API Documentation
   a. GitLab API
      i. The GitLab API enables us to display real-time data on commits, closed issues, and unit tests created by each site maintainer. We fetch issues via the https://gitlab.com/api/v4/projects/[PROJECT_ID]/issues endpoint. For each person, we count the issues assigned to them that are currently closed. Commits are retrieved through the https://gitlab.com/api/v4/projects/[PROJECT_ID]/repository/commits endpoint, where each commit associated with a person's email address contributes to their commit count.
      ii. Since GitLab's API support for unit tests is limited, tracking the correct number of unit tests per person involves a partially manual process. Contributors include the phrase "n unit tests" in their commit messages to denote the number of tests added. Each such message increments the contributor's unit test count accordingly.
      iii. All API calls to GitLab are made asynchronously. Given GitLab's limit of 100 items per page, we repeat API calls for each subsequent page, using the page query parameter to specify the desired page of data. We set the per_page parameter to 100 to maximize the amount of data received per call. For example, retrieving the second page of issues would involve a request to https://gitlab.com/api/v4/projects/[PROJECT_ID]/issues?per_page=100&page=2.

   b. Google Maps API (RESTful)
      i. We have integrated Google Maps to provide location-based services and interactive maps for the Housing and Childcare models, enhancing user navigation.
   c. ParentAidAtx API (RESTful)
      i. Our API utilizes GET calls to retrieve information about all models on the website, typically based on their attributes. This functionality allows users to either obtain all instances of a

model at once or find a specific instance, depending on the nature of the call. For more detailed information, please refer to the API documentation available at https://documenter.getpostman.com/view/42442568/2sAYdZst Bv.

6. **Tools**
   a. **Node.js:** Used for package management. Run 'npm install' to install the required dependencies from the package.json file.
   b. **React:** Frontend framework for building user interfaces
   c. **Bootstrap:** CSS framework for responsive styling and UI components
   d. **AWS Amplify:** Website hosting
   e. **Postman:** API Documentation
   f. **Selenium:** For web scraping
   g. **Flask and Flask-CORS:** For the API endpoints
   h. **Gunicorn:** Python WSGI HTTP Server
   i. **SQLAlchemy:** To manage the database
   j. **Railway:** For hosting PostgreSQL database
   k. **AWS EC2:** For hosting the backend API

7. **Frontend Hosting**
   a. Our website is hosted and deployed with AWS Amplify. AWS Amplify ensures a stable user experience and allows our site to scale. Furthermore, with AWS Amplify, the website will automatically update when a push is successfully made to GitLab. The domain parentaidatx.me was obtained from Namecheap.
8. **Backend Hosting and Database Structure**
   a. The backend Flask API is hosted with AWS EC2. The backend PostgreSQL database is hosted on Railway. Within the database, there are three tables—one for each model—to hold the data we gathered, either from web scraping or using existing APIs. Our FlaskAPI endpoints then serve the data from the database to be used in our frontend. The images below show our Flask API in AWS EC2 and PostgreSQL database tables in Railway. The API is accessible at https://api.parentaidatx.me
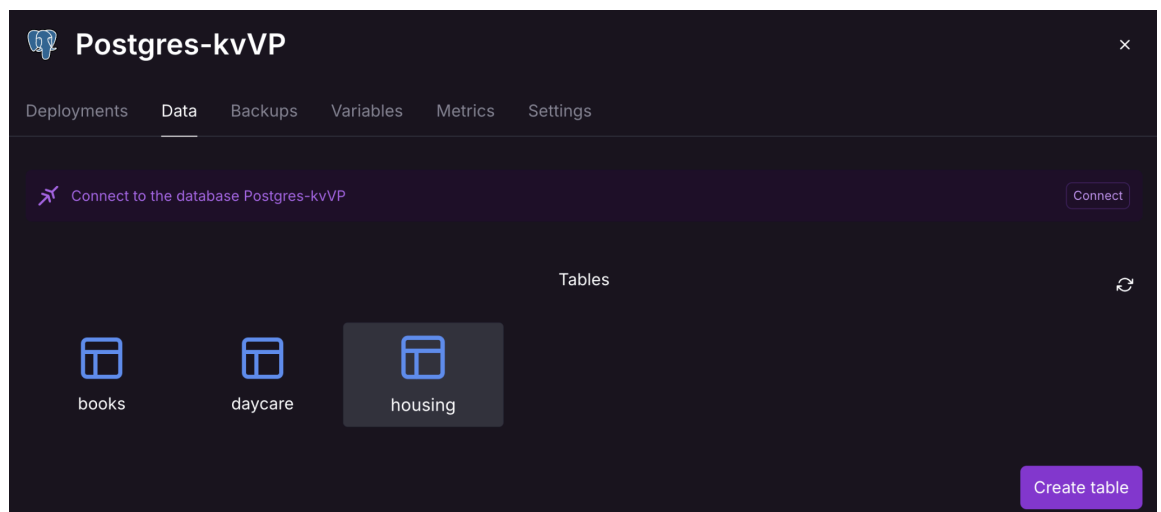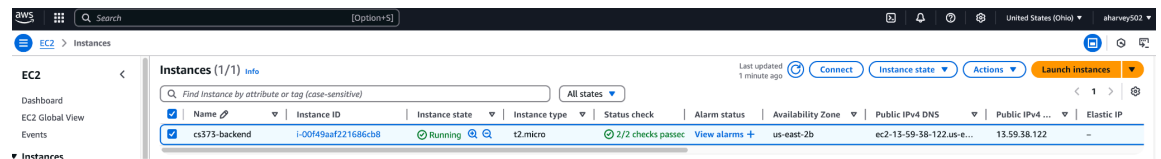
9. **Architecture**
   a. **System Components**
      i. **Frontend**
         1. The frontend of ParentAidATX is built using React, HTML, CSS, JavaScript (JSX), Node.js, and Bootstrap. Each page features a navigation bar created by the App.jsx and main.jsx files. All frontend files are located in the frontend folder.
      ii. **Backend**
         1. The backend uses AWS EC2 to host the Flask API and Railway to host the PostgreSQL database.
      iii. **CI/CD Pipeline**
         1. The project is integrated with GitLab's CI/CD pipeline, allowing for continuous integration and deployment. This setup ensures that any code changes are automatically tested and deployed.

   b. **Folder Structure**
      i. **Frontend Folder**
         1. Contains all the frontend files, including the main components (App.jsx, main.jsx) that create the navigation bar.
         2. **frontend/__tests__:** Contains the frontend tests.
         3. **frontend/public:** Stores all images used in the application.
         4. **frontend/src:** Stores all the models utilized by the frontend components.
         5. **frontend/src/components:** Contains the Navbar component, Cards components for each page, and the pagination component.
            a. **frontend/src/components/pagination.jsx:** Added in phase 2, this allows users to navigate through pages and displays a limited number of visible page links with ellipses for better usability.
      ii. **Backend Folder**

1. **backend/app:** contains all the backend files related to our application
    a. **api.py:** Our Flask API. Contains endpoints to serve data we gathered for each of our models.
    b. **childcare_scraper.py:** Web scrapes https://mybrightwheel.com/ to get daycare (childcare) data and then inserts that data into the database "Daycare" table.
    c. **books.py:** Utilizes the Google Books API to get books and then inserts that data into the database "Books" table.
    d. **housing.py:** Utilizes the Google Maps API to get housing and then inserts that data into the database "Housing" table.
    e. **Dockerfile:** Defines a containerized environment for the backend application necessary for hosting the backend in AWS EC2. When executed, it uses gunicorn to serve the API on port 5000. Ngix was then used to make it HTTPS.
    f. **postman_tests/:** Contains API tests.
    g. **unit_tests/:** Contains backend tests.

c. **Data Attributes**
   i.   Each instance within the system has five attributes and includes at least two sets of media, ensuring comprehensive data representation.

d. **Data Flow**
   i.   Data flows seamlessly through the system, from collection and processing on the frontend, to storage and retrieval on the backend.

e. **Communication Protocols**

i. The system uses REST APIs for communication between the frontend and backend components, ensuring smooth data exchange and interaction.

f. Deployment

i. The system is deployed and maintained using GitLab's CI/CD pipeline, enabling continuous integration and deployment. This ensures that updates and new features are rolled out seamlessly without disrupting the user experience.

10. Challenges

    a. Backend Hosting

        i. There was a lot of trial and error with backend hosting. We initially used Railway to host the entire backend API AND Database, but we were told by Professor Downing that we had to use AWS EC2 to host the API, so now we are doing that.