EDEN PRAIRIE PLAYGROUNDS WEB MAP APP

Project Plan by Brian Strock – GEOG 778 – Spring 2022

OVERVIEW

AUDIENCE

Families in and around Eden Prairie, Minnesota who seek to discover and enjoy outdoor public recreation spaces based on the availability of playground amenities.

INTENTION

The app experience promotes the discovery and accessibility of outdoor amenities in the City of Eden Prairie, with features specifically tailored for the needs of playground and public space users, such as:

- Locating playground sites which often do not appear in GPS app searches (Google Maps)
- Identifying playground spaces from the surrounding park environment
- Navigating directly to playground via georeferenced XY location, rather than GPS app addresses, which often lead to other areas of a larger public space far from playgrounds
- Filtering available parks based on available equipment type or nearby amenities
- Allowing park users with accessibility needs to find outdoor spaces which they find enriching

GEOGRAPHIC EXTENT

City of Eden Prairie, MN, specifically featuring Eden Prairie's 26 parks which offer playgrounds.

DELIVERABLES

This project will develop a modern web map app called Eden Prairie Playgrounds, designed to help families in my city of Eden Prairie, MN who want to locate fun playgrounds to visit. The final deliverable will feature a modern tech stack based on open-source, open-web technologies. This will ensure that the end-user experience is available on any device or platform, to provide access to the broadest possible audience.

The app will feature an interactive web map featuring a custom-generated dataset providing an inventory of specific playground fixture types. A user interface component will allow for attribute- and spatial-based queries, as well as links to turn-by-turn routing. Users will be able to create accounts to submit playground reviews and report problems, as well as save Favorites.

DEVELOPMENT AND DELIVERY

The implementation will be based on a client-server architecture built on a relational database management system, which will be queried and updated via a custom REST API with both public and authenticated endpoints. These endpoints will be accessed via a front-end website, built using a Javascript frontend framework.

Development will be tracked and versioned via GitHub. Since a dev prototype for the API already exists, backend development will focus on staging the API via a Heroku deployment integrated with a GitHub commit trigger. From there, the frontend will be reworked into a modern JS Framework, and additional features will be implemented.

The source code will be published under an open-source license, so that other users can implement similar applications for their communities.

DATA ELEMENTS AND SOURCES

ELEMENT	TYPE	SURFACE	SOURCE
Basemap	Slippy Web Map	Map window	Mapbox
Playground Addresses	Table (db.sites)	Geocoding, Site Callout	City of Eden Prairie
Playground Polygons	Polygon	Basemap	Georeferenced
Playground Points	Point	Map Marker	Polygon centroid
Playground attributes	Database tables	Query Surface, Site Callout	Developer site surveys
User favorites	Point	Alternate Map Marker	Table (db.users)
User reviews	Text, Image (Stars)	Site Callout	Table (db.reviews)
C. of Eden Prairie Border	Line	Basemap	C. of EP GIS resources
User Location	Point	Basemap	Client (if provided)
POI	Search Box/Point	Map window	???
Search Radius	Polygon	Basemap	User location or POI

TECHNOLOGY STACK (GRAPHIC)

Application Layer	Tech Stack	Components

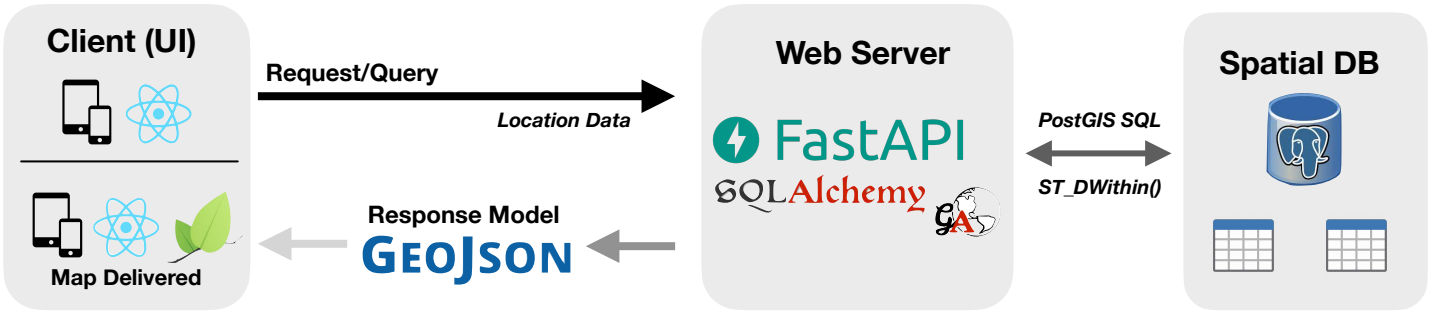
DATABASE
- *Data storage for:*
  - Park/playground attributes
  - Geospatial data
  - User accounts
  - User reviews/reports
	PostgreSQL + PostGIS	- **PostgreSQL**: DBMS
		- **PostGIS**: Spatial DB

API (Python 3.9)
- *Web server as interface layer between client and database*
  - Dynamic park/playground attribute querying
  - User account creation
  - User authentication
  - Informing map popup data
  - User review submission
	FastAPI, SQLAlchemy, GeoAlchemy, GeoJSON	***Python Packages used:***
		- **FastAPI**: REST Endpoints
		- **SQLAlchemy**: Database ORM
		- **GeoAlchemy2**: Spatial DB
		- **Pydantic**: Data models
		- **GeoJSON**: Response data

USER INTERFACE
- Provides users with
  - Prominent map featuring playground locations
  - Interactive map popups to explore park information
  - Spatial- and attribute-based querying of playground data
  - User review submission
	React + Leaflet	- **React**: UI Framework
		- **Leaflet**: Map interface

Brian Strock - GEOG 778 - Spring 2022

SERVICE DELIVERY MODEL (GRAPHIC)



TECHNOLOGY STACK (DETAILED)

DATABASE COMPONENTS:

POSTGRESQL/POSTGIS

- Used to define relational database schema and host/manage project data
- PostGIS provides spatial data storage and high-efficiency spatial indexing/querying

API COMPONENTS (PYTHON 3.9):

FASTAPI (REST ENDPOINTS/ASYNC WEB SERVER)

- Highly intuitive development patterns for endpoint management
- Async implementation provides high availability from small deployment scope
- Uses Pydantic data structures/type hinting for defining queries and responses

PYDANTIC (DATA MODELING)

- Used to define custom data structures using Python type hints
- Integral to FastAPI's JSON integration and responses
- Offers methods to convert between Pydantic models and ORM objects

SQLALCHEMY (OBJECT-RELATIONAL MODEL, DATABASE INTERACTIONS)

- Seamless converted between FastAPI query structures with dynamic content and SQL queries, the results of which are converted to GeoJSON responses
- ORM-based model provides high-efficiency database operations, including implicit autobegin behavior to manage transactions and rollbacks
- Bound object behavior/query parameterization prevents SQL Injection
- Clear syntax and exceptional documentation/maintainability
- Async PostgreSQL drivers work together with FastAPI for high availability

GEOALCHEMY2 (PYTHONIC SPATIAL DATABASE BINDINGS)

- Provides PostGIS datatype bindings to SQLAlchemy's declarative interface
- Extends PostGIS SQL server functions to the SQLAlchemy Python interface
- This in turn provides highly efficient server-side spatial querying
- Query results objects can be easily serialized into GeoJSON

UVICORN (ASYNC WEB GATEWAY)

- ASGI gateway allows for scalable async webserver hosting

USER INTERFACE COMPONENTS

REACT (UI FRAMEWORK)

- Allows interface to be built using independent state-based components
- Virtual DOM model faster than rendering directly in traditional DOM
- Enables more complex application architecture while simplifying code structure

LEAFLET.JS (WEB MAP COMPONENT)

- Open-source web map technology with flexible basemaps and lots of extensions
- Offers interface for generating point/line/polygon layers, as well as custom map markers and interactive callouts
- React-leaflet bindings allow Leaflet components to be rendered as React components

DEVOPS/HOSTING COMPONENTS

GITHUB (VERSION CONTROL)

- Allows for code versioning and remote backup management
- Open-source platform creates visibility and promotes project/developer
- Codebase can be open-sourced to allow other users to generate similar apps for their localities by customizing location attributes and generating basic datasets

HEROKU (DATABASE AND API HOSTING)

- Free small-scale PostgreSQL hosting (10k row limit well above project requirements)
- Free app hosting via Docker container definition
- Github integration allows for seamless CI/CD deployment such that code pushes can trigger app rebuild and deployment automatically

GITHUB PAGES (WEB HOSTING)

- Free, reliable hosting with zero-code deployment and easy integration with project
- Can attach site to custom domain if desired
- Analytics implementation allows for site performance monitoring

SECURITY ARCHITECTURE

SECRETS HANDLING: API/DATABASE

The API app uses Docker for build definition and passes connection string secrets into the container at build time via a Secrets Key Vault in Heroku. For development, sensitive credentials are stored in local environment variables and are never checked in to version control in code.

USER DATA

For account creation, this app requests an email address, first and last name, and a password.

SECRETS HANDLING: USER CREDENTIALS

Upon account creation, users are prompted with a short form requesting the data above. The submission is sent via HTML Form data. The API then receives the submission and hashes the user's password using bcrypt[SHA256]. The hash is then stored in table db.users.

SECRETS HANDLING: AUTHENTICATION

Upon logging in, user login form submission data is sent via HTML Form. The API receives the submission and hashes the submitted password. The submitted email is used to query db.users for the previously stored password hash, and the hashes are compared.

If hashes match, the API returns an HTTP Authentication Header containing an OAuth2 Bearer Token implemented via JSON Web Token (JWT). The client then stores this token for the duration of the user's session, and the token expires after some time.

BUDGET

I expect to spend 10 hours per week on development and testing work for the next 10 weeks, for a total of around 100 hours of dev time. The time breakdown is expected to be as follows:

40% overhauling the UI
20% testing
20% refactoring and documentation
20% API expansion and deployment

This project is expected to have no costs aside from time, due to using a 100% FOSS tech model.

GOALS AND MILESTONES (TIMELINE)

	By 2/21	By 3/7	By 3/21	By 4/4	By 4/18	By 5/2
Goal	UI design and tooltip layout complete	API and database staging completed	Functional Webmap and basic UI elements in place and working	Query sidebar component and POI searching fully functional	User account creation, login, reviews, and favorites all working	Move to production and submit final demo
Milestone	Wireframes and interface layout map	Front-end components pull data from live API instead of localhost	Leaflet webmap, menu bar, sidebar, and playground tooltip as rendering as components	Can search for POI or user location on map and see nearby results	User can execute end-to-end workflow in live web app	App is live, turn in Executive Summary

DATABASE SCHEMA (GRAPHIC)