Alex Zarley
Geog 777

Project 2 Report

I chose to create a web application for visitors to Petra Archaeological Park in Wadi Musa, Jordan. In my previous work as an archaeologist, I spent 4 summers excavating in the park and always thought an interactive map and a mobile app could be helpful for visitors to the park. The Archaeological Park has been a UNESCO World Heritage site since 1985 and encompasses the ancient Nabataean city of Petra and its surrounding area. There is a main path through the park that is sandy, but relatively flat, and allows you to see many of the sites in the park. There are also miles of trails branching off this main path and spreading throughout the park's mountainous terrain that allow you to see some of the more spectacular places in the area, such as Jabal Haroun (Aaron's Tomb), The High Place of Sacrifice, and Al-Khazneh (The Treasury). You could spend a week at Petra and still not be able to explore all it has to offer. The goals in creating this app are to help users plan their visit to Petra, give them some easily accessible additional information about the sites they are seeing, easily orient themselves with their location in the park, report issues they see on the trails, stairs, and bridges of the park, and be able to more easily identify where they can find refreshments, food, and restrooms during their visit. My application stack uses JavaScript, Leaflet, and Pug on client-side and Express, Node.js and PostgreSQL on the server-side.

This app is designed to be used in a browser to explore Petra and its sites before visiting the park, as well as on mobile devices while in the park. I created all the data layers. The locations within the park and names of the historical sites are accurate. The location of the points of interest and trails are also accurate. I made up some of the dates and eras of the historical sites, but I've included external links for background information on each site.

My four data layers displayed on the map are the park boundary (polygon), the paths and trails through the park (line), points of interest (point), such as restrooms, restaurants, and visitor information, and the historical sites of Petra (point). My fifth layer is the user-submission layer, and it allows users to report issues within the park. This layer is not displayed on the map because it would clutter the display, but a marker is added to the map at the location the user selects for the issue.

The layout of the application is a menu bar across the top of the screen that holds the application title, a search box, a filtering accordion widget, a button to submit an issue a user has encountered in the park, and a button to create a marker icon and zoom/pan to the current location of the device. The map takes up the rest of the screen and opens centered on the park. The map is mobile friendly in that as the window size shrinks, as it would on a mobile screen, the widgets in the top menu begin to stack rather than stay off screen. The map will also automatically zoom out and recenter as the window size shrinks. The button to identify a user's current location will use the geolocation capabilities of the user's device.

The five functions I included in my application are explore, search, filter, submit, and locate. For explore, you can click on any of the icons and a popup appears with more information about the historical site or point of interest. The historical site popup includes a little information about it and a link to an external site with more information. Search allows you to search for any of the included historical sites by name. The names used in the auto-complete of the search are dynamically loaded from the results of one of the server-side queries. This means as any more sites are added to the database, they will automatically be included in the auto-complete search. When you search for a site, the map pans and zooms to the selected site, and the popup appears. Filter allows you to filter the historical sites by the era they were in use. The three main eras represented in the sites of Petra are Nabataean, Roman, and Byzantine. There are three checkboxes, one for each era, that are selected by default. As you uncheck an era, sites from that era are removed from the map. They are added back when you check the era. The filtering is done on the client-side. As any checkbox changes, I retrieve the values from each checked checkbox, and then use the built-in Leaflet filter method to filter the features in the sites layer based on which are checked.

The submit function allows users to submit issues within the park. When the ‘Report an Issue’ button is clicked, a popup appears with instructions for submitting an issue. There is a dropdown menu to select the type of issue. These values are loaded from the results of a query retrieving all records from the issue_type table in my database. As more issue types are added to the table, they will automatically be added to this dropdown. The next element is a checkbox that allows you to choose the location of the issue on the map. After clicking the button, you can click any point inside the park boundary and a marker icon will appear. There is also a textarea element that allows users to submit comments about their issue. The comments column in the database can store 255 characters, and the textarea is limited to 255 characters. When the user clicks the submit button in the popup, I retrieve the selected value in the issue type dropdown menu, the longitude and latitude of the marker point on the map, and any comments added to the textarea and send them to the server with an ajax PUT request. The insert sql query is stored on the server, and I use string interpolation to populate the values in the sql query with the values passed from the client. The record is then inserted in the issues table in the Postgres database, and the result of the query is returned to the client and displayed in a popup message, which lets the user know if their issue was successfully submitted or not. If a point is not selected on the map or an issue type is not selected, a user cannot submit an issue. When the user closes the “report an issue” popup, the marker point on the map is removed.

The locate function takes advantage of geolocation capabilities of the user’s device. When the button is clicked, I attempt to retrieve the latitude and longitude from the device. If geolocation is disabled, a popup appears letting the user know this and that their location cannot be detected. If geolocation is enabled, a marker point is created at the user’s location and the map zooms/pans to that location. A ‘clear marker’ button, that is hidden by default, also appears after the locate button is clicked. When the user clicks the ‘clear marker’ button the button disappears, and the marker is removed from the map.

I chose to use a Postgres database with Node.js for the back end of my application. All data for the application is stored in the database and is retrieved with JavaScript via put and get requests with Express routing. All SQL queries to retrieve and insert data in the database are stored on the server-side. When the app initially loads in the browser, I query the database to retrieve attribute and geometry data as GeoJSON’s for each of the four map layers. I also retrieve the attribute data for the four lookup tables in the database and return those results as JSON’s. Each of the eight query results is added to an array which is then passed to the client.

The client-code is written in Pug and JavaScript. Pug looks like HTML with many characters stripped away. I use it to structure the web page and its elements and add links to stylesheets and JavaScript libraries I use. I created the map and its functionality with Leaflet and jQuery. After receiving the map data and lookup tables from the server-side requests, I create GeoJSON Leaflet layers, style them, add popups, and bind event listeners. All map icons were created with Geoapify, using the Material Design Icon and Font Awesome icon libraries.

The ER-Diagram (Fig. 1) and Logical Schema (Fig. 2) appended to the report show the database design. There is no explicit spatial relationship among the geographic datasets. The park_boundary and trails tables have no relationships with other tables. The points_of_interest and issues tables each have one related lookup table. The sites table has two related lookup tables.

ER Diagram

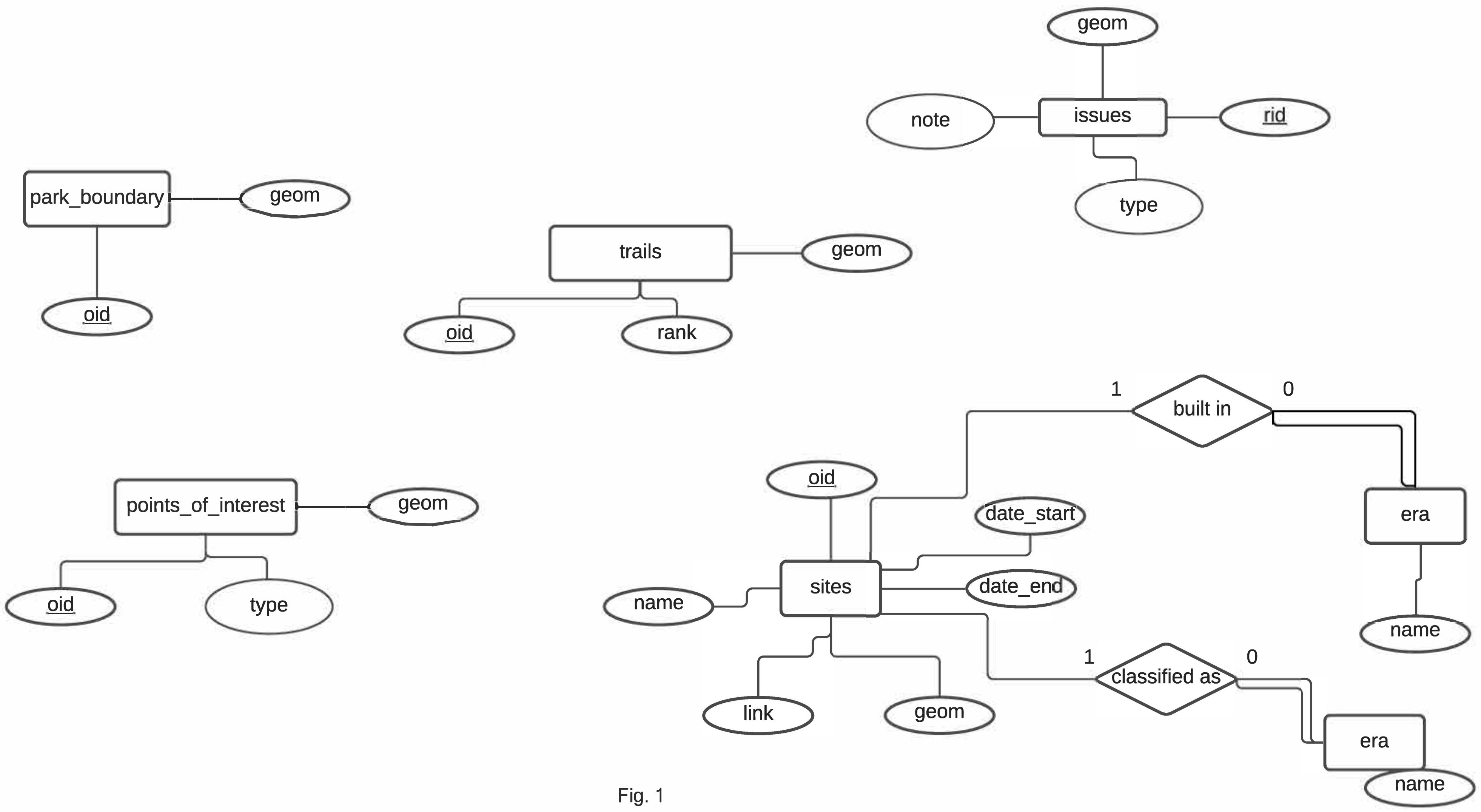

Fig. 1

Logical Schema

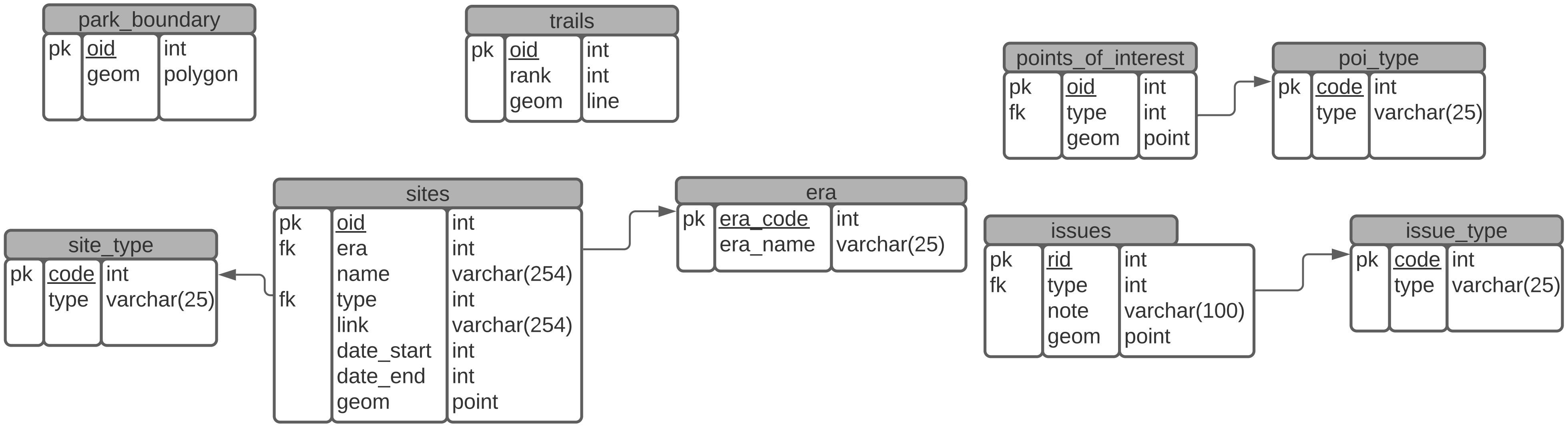

Fig. 2