Project Report - Team009 “e”

Michael Vilsoet mvilso2, Franklyn Wu fewu2, Shreya Basu shreyab4, Alex Desjardins aed6

1.

The UI of the web application is the main deviation from the original proposal. After beginning the project, we understood more about our database and what was possible with it. For example, it was planned that the ‘Activities’ (originally titled ‘Things to Do’), ‘Amenities’, and ‘Events’ pages would only display relevant park-specific information once the user selected a Park. Instead, all 3 pages display all available activities, amenities, and events, respectively at all parks. Users can then use the specific form on each page to filter by park name, state, keywords, etc. There has also been the addition of two new pages: ‘Parking Lots’ and ‘In Season’. The ‘Parking Lots’ page contains a map plotting all the available parking spaces at various national parks and their availability. The ‘In Season’ page allows users to filter by season and find out how many ‘Activities’ are available at a given park during that season. We feel like our deviation from the mock-up has included only improvements to our project and that users will always leave the site satisfied regardless of which feature they decide to try.

2.

We believe the application successfully achieved the project’s original purpose of being an extremely streamlined and informative reference tool for all the services and locations operated by the U.S. national park service. Although the information we were using is all publicly available, there are a lot of bells and whistles on other sites such as Tripadvisor or the National Park Service website itself that means it can be difficult for people to gather and compare a large amount of information at once. Our site gathers a large amount of information on different aspects of visiting national parks with very intuitive search options on a site that is extremely easy to navigate. We believe it does a good job at serving as a jumping-off point for people who are in the early stages planning trips, since it’s easier to use if you’re not necessarily sure on what you’re looking for when compared to a more detailed website. However, due to time constraints there were naturally a lot of more ambitious features that we wanted to implement that weren’t viable in the project timespan. These include the ability to apply filters to a previous search query, and the ability to view activities and parks on a map similar to the parking lots tab in our final version. These quality-of-life improvements would have increased usability by improving data visualization and giving users more control over search results.

3. 4.

As outlined in the initial proposal, the data was successfully retrieved from several calls to the National Park Services API. A script makes about 30 calls to the API which cycles through each event and updates the Events table after properly formatting each entry. Compared to the original database design proposed by the ER diagram, there were only notable changes primarily affecting the Events table schema. First, the startdate and enddate were converted to a date data type; this is a more appropriate representation of data contained in both of these fields and made it easier to build the trigger that ensured newly added events did not have an end date earlier than the current date. We also realized that it is simply proper database design to use the "date" variable type for date columns to increase the amount of useful queries that can be made on any given table.

5.

As mentioned above, the amenities and activities information was moved to separate pages for a cleaner UI. This design decision resulted in significant changes to the functionality. Instead of filtering the table by activities they'd like to participate in, users can take advantage of a text-based search. To retrieve park-specific information about activities they would like to participate in, users would navigate to the Activities and use the necessary search filters available on that page. Also, the landing page still contains a table listing all parks in the NPS system, but some of the proposed fields were removed, specifically *operating hours* and *entrance fee costs*. This was done because it was determined that these details are readily available on the park website that's hyperlinked for each park entry. Lastly, the creative function integrated in the web application allows for users to locate the parking lots of any National Park that they choose. This bridges the gap between simple web application and real-life usage. The function is an interactive map that uses LeafletJS to give users a physical location to drive to so that they can *actually* visit the National Park. We feel like this is one of the most important aspects of our application because it provides a very attractive and tangible view of the National Parks which could inspire our users to visit the parks. More importantly however, it would increase the likelihood of repeat visitors to our application, compared to our initially proposed Park Description Generator.

6.

The trigger in our database schema is a very important quality-of-life improvement for our application. The basic functionality of the events page is that we can add and edit events for any park that we choose. However, it also allows the user to enter any data that they want. To prevent users from submitting events with incorrect details, we created a trigger that would clean up all submissions to the events page. When a user submits the form for an event on the website, our "before insert" trigger is activated and ensures that every event has a proper title, description, and a start/end date that hasn't already passed.

The stored procedure in our database is a smart suggestion tool for users. It finds parks that are holding events which are occurring in the next week. To make it even more helpful, it also searches for free parking. Then, it returns parks that are "happening" which have more than 3 events that fit the criteria. It utilizes a cursor, a row-by-row loop, a temporary table, and several complex queries within it to provide users with fantastic park suggestions on the home page. We feel like this is the main attraction for users that want to make a fast plan to visit the National Parks.

In Season is an advanced query that requires the theta join of Activities and Park and filters out free activities. It returns a list of parks alongside the amount of free, in-season activities that are offered. “In-season” is determined by user input. This advanced database program complements the application by providing users a perspective on which specific park might have the highest value at a given time of year. You can look up specific activities in the “activities” tab, but seeing ALL the activities offered by a park at a given time of year might be more relevant in terms of determining an overall destination. We filter by activity fee to help our users who are traveling on a low-budget.

Finally, we included an advanced query which joins the Events table and the Parking Lots table. This query is useful for users because it allows them to find events that have free parking lots. Our users will never pay more than they have to. The query is manifested as a green check mark next to the park which is hosting the parking lot.

7.

(Frank) I personally struggled most with development involving the API, i.e. communication between backend and frontend. It was a skill I had very little experience in and had to learn very quickly from scratch. I believe that if I had more experience in this aspect of development, I could’ve implemented far more ambitious features and functionalities that I unfortunately just didn’t have time for. I felt HTML + CSS alone was relatively straightforward to learn, but figuring out how to dynamically send requests to the database and subsequently update the front-end in real time was frustratingly complex and time-consuming. Due to the focus on databases in class, I found the SQL queries the easiest part to write, but the query’s complexity was limited by what could be implemented in the frontend. I would recommend that people without web development experience invest a lot of time into self-teaching API development.

(Michael) The biggest challenge that we faced was learning HTML + CSS from scratch. After being given some sample code by the TA, we were dropped in the deep end and told to make a website. Most of us had never operated a frontend framework before, so doing this project was often simply googling tutorials on HTML. It was important that we each communicated with each other what we have learned so that each member could benefit from each tutorial we used. Finally, keeping up with HTML formatting is very difficult in that the code will become very cryptic and difficult to read unless we make proper dividers and indentation. We only realized this halfway through the development and it was a bit of a struggle to reorganize our HTML code.

(Shreya) It was a huge challenge to tackle the third-party frameworks and packages that are used in our project. For example, we spent several hours trying to figure out how to use Ajax to send queries to our database. Ajax has unforgiving syntax and the error messages are not always clear, especially since none of the other members had prior experience with this framework. Once we figured out one query, however, we were able to copy the syntax for every other query. Other than that, the LeafletJS library was yet another library with its own syntax that we had to learn in order to implement it into our project. Overall, it is of utmost importance that each group member is willing and able to learn new types of programming quickly and doesn't get discouraged or distracted during the process. It can be very rewarding to learn all these different skills!

(Alex) Another major technical challenge that we faced was accessing and populating the event modal upon adding or editing an event from the table on the events page. Accessing the data we were able to do using html forms; however, populating the event modal with existing information was more difficult. The desired functionality was that when the edit button was clicked for a particular event, the event modal would pop up and contain that event's relevant information. We tried storing the event json in the content field of its corresponding event modal; however, the event modal of each event has the same id, so when we tried finding the event modal in the js script, it would always access the information of the first event. Eventually we discovered that the event json could be stored in the content field of that event's button and then be accessed in the js script using click events. Also, since the only data accessible through the modal directly is that of the first event and not the current event (due to duplicate ids), we found it necessary to store the event id in a global field in the js script so that the proper event id could be identified after the modal had popped up. Otherwise, when submitting the event data changes in the modal, since event id is not a field visible to the user, we would not be able to access the event's id (since it is inaccessible through the modal) and therefore not be able to determine which event to update in the database.

8.

No other key changes have been made since the original proposal.

9.

There is some room for improvement completely unrelated to the interface. Our two advanced queries were pretty well-optimized due to using an index that greatly reduced the cost of their joins. However, the smart suggestion tool, (which was the stored procedure), could also use a lot more optimization, since it has a high cost and high real runtime. As noted during the demo, if we were to use a noSQL database, the smart suggestion tool would benefit significantly since in its current state it requires a lot of resource-intensive joins. The trigger, if given more time, could be improved to be more comprehensive and robust in filtering out bad submissions when interacting with the create, update, delete (CUD) features. Given more time we would also implement security features around CUD features, allowing for control over who is authorized to change what information and how. This is because in a real-world setting some users should have more authority to change information than others. That being said, due to the fact that our group's biggest difficulties were related frontend and API development, a lot of our SQL queries and backend development were undoubtedly limited by the interface- since we had to consider whether we could implement an intuitive way for the user to interact with the backend feature. Without the constraints of the interface, we would have implemented more complex and dynamic queries, allowing for more accuracy, precision, and flexibility in the user's search parameters, (and consequently in the information displayed).

10.

We evenly split the work into four parts and each followed a timeline to complete prerequisite requirements first. For example, at stage 5, we could branch off our tasks onto multiple landings on the website. Each person worked on one aspect of the frontend which allowed all four of us to learn more about Flask. Over break, some of us got a bit more work done so that the following week would be easier for the next few people and the proper infrastructure would be in place to actually complete final tasks such as trigger and stored procedure creation. For stages 1-4 and the final stage, we simply met up as a team and all did work until the stage was done. In these cases, we usually met up at either CIF or Everitt at a pod where each member could communicate and meet. Finally, we had a group chat to stay in touch with each other and even delegate tasks outside of meetings.