Final Project Documentation

Student's Name: **Hamad Alnuaimi**

Course Name: **IEE 305 Information Systems Engineering**

Professor's Name: **Emmanuel Miguel Gonzalez**

Assignment Date: **Dec 9,2025**

1. Introduction

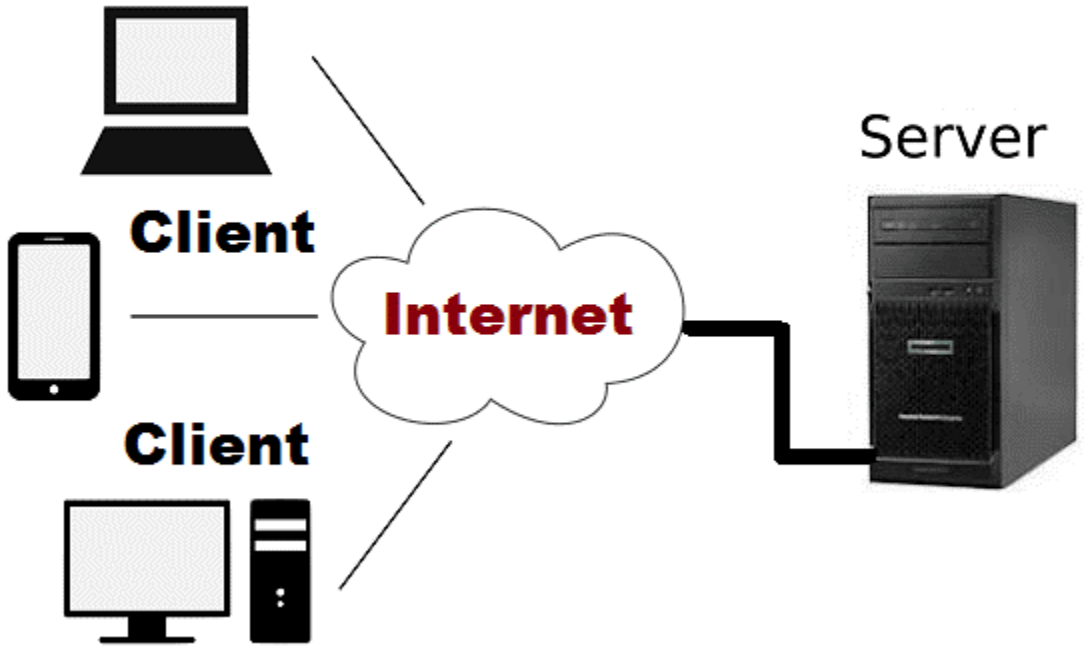
National Park Service (NPS) coordinates different operations such as visitor services, scheduling park activities and maintaining facility across numerous parks and visitor centers. The staff members must confirm that services provided meets visitors' demands while balancing safety and budget constraints. Presently, the data about operations at the park, facilities available and number of visitors is scattered and therefore cannot be used for optimal decision-making. The managers have currently been using data from manual spreadsheets, exposing them to underutilized resources, missed opportunities and inevitable risks. Due to this, the project aims to implement a standardized and fully functional application interface to ensure accuracy transparency in park operations.

The National Park Service API implemented will provide data about park facilities, designations, and visitor activities. The managers at the national pack will embrace this API in evidence-based decision making. It will provide factual information that the managers can rely on rather than fragmented reports when making decision about park operations. Also, they will utilize real-world visitor data for capacity planning since the API will incorporate diverse metrics like defective usage, visitors number and activity participation. The engineers will analyze API patterns to explore expected demand, approximate peak load, and make future predictions. Finally, the API will help prioritize staffing and park maintenance based of usage and activity trends because it evaluates frequency of events, facilities, and trails.

2. System Design

2.2. Client-server architecture diagram

The system follows a **client–server architecture**, where data access and business logic are centralized on a backend server and consumed by clients via a RESTful API.



2.3. Final ER model and relational schema

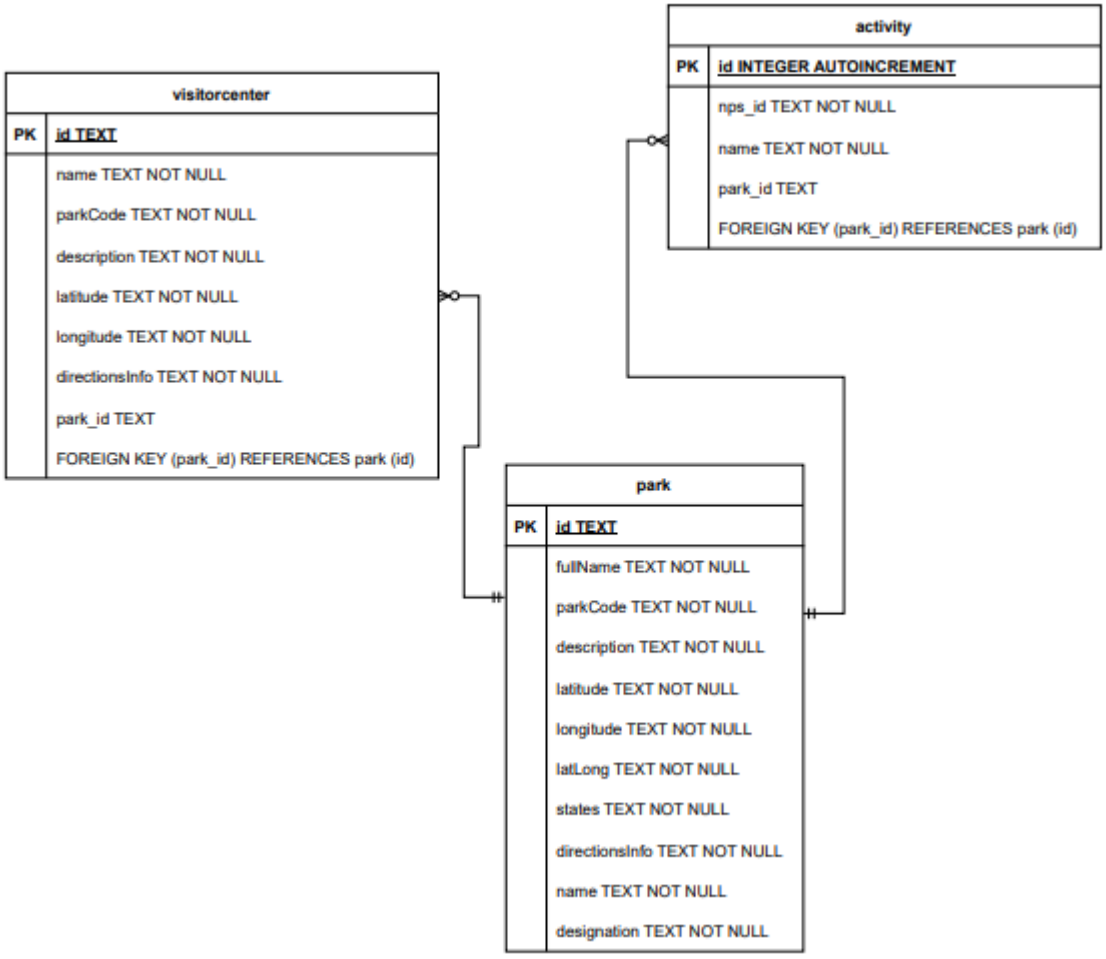
Entity–Relationship (ER) Model

The final ER model consists of **three core entities**, based on National Park Service (NPS) data:

1. **Park**
2. **VisitorCenter**
3. **Activity**

Relationships:

- One **Park** can have **many VisitorCenters** (1:N)
- One **Park** can have **many Activities** (1:N)
- Each VisitorCenter belongs to exactly one Park
- Each Activity record represents an activity offered by a specific Park



2.4. API endpoint documentation

I have created 8 API endpoints in the backend.

Park Endpoints

Endpoint	Method	Description
/parks	GET	Retrieve a list of all parks
/parks/{park_id}	GET	Retrieve a specific park by ID
/parks/{park_id}/visitor-centers	GET	Retrieve visitor centers for a specific park
/parks/{park_id}/activities	GET	Retrieve activities offered at a specific park

Visitor Center Endpoints

Endpoint	Method	Description

/visitor_centers	GET	Retrieve all visitor centers
/visitor_centers/{visitor_center_id}	GET	Retrieve a visitor center by ID

Activity Endpoints

Endpoint	Method	Description
/activities	GET	Retrieve all activities
/activities/{activity_id}	GET	Retrieve an activity by its ID

2.5. Technology choices with justification

I use FASTAPI for building the backend due to its simplicity to use as well and its performance.

I also used SQLModel(ORM) as it combines SQLAlchemy and Pydantic into a single, unified model

The database used was SQlite and it is easy to get it working without having to install and configure many components.

3. Database Implementation

3.1. Schema definition

3.1.1. Create Park table

```
CREATE TABLE park (
   id TEXT PRIMARY KEY,
   fullName TEXT NOT NULL,
   parkCode TEXT NOT NULL,
   description TEXT NOT NULL,
   latitude TEXT NOT NULL,
   longitude TEXT NOT NULL,
   latLong TEXT NOT NULL,
   states TEXT NOT NULL,
   directionsInfo TEXT NOT NULL,
   name TEXT NOT NULL,
   designation TEXT NOT NULL
);
```

3.1.2. Create visitor center table

```
CREATE TABLE visitorcenter (
   id TEXT PRIMARY KEY,
   name TEXT NOT NULL,
   parkCode TEXT NOT NULL,
   description TEXT NOT NULL,
   latitude TEXT NOT NULL,
   longitude TEXT NOT NULL,
   directionsInfo TEXT NOT NULL,
   park_id TEXT,
   FOREIGN KEY (park_id) REFERENCES park (id)
);
```

3.1.3 Create activity table

```
CREATE TABLE activity (
   id INTEGER PRIMARY KEY AUTOINCREMENT,
   nps_id TEXT NOT NULL,
   name TEXT NOT NULL,
   park_id TEXT,
   FOREIGN KEY (park_id) REFERENCES park (id)
);
```

- park.id is the primary key (string), that I took directly from the NPS API park ID
- visitorcenter.park_id is a foreign key to park.id and this is used to establish a one-to-many relationship (one park -> many visitor centers).
- activity.id is a local integer primary key, while activity.nps_id stores the NPS activity identifier
- activity.park_id links each activity record to a specific park (one park -> many activities).

3.2. Normalization analysis and functional dependencies

3.2.1. Park

-the primary key is the id

Functional dependency:

id -> fullName, parkCode, description, latitude, longitude, latLong, states, directionsInfo, name, designation

All non-key attributes depend directly on the key and not on each other.

There are no repeating groups or partial dependencies and no transitive dependencies involving non-key attributes.

The relation Park is in 3NF

3.2.2. VisitorCenter

-the primary key is the id

Functional dependency:
id -> name, parkCode, description, latitude, longitude, directionsInfo, park_id

park_id is a foreign key, but not part of the primary key. All attributes are fully functionally dependent on id.

visitorcenter is also in 3NF, as there are no partial or transitive dependencies on a composite key.

3.2.3. Activity

The id is the primary key

Functional dependency:

id -> nps_id, name, park_id

All non-key attributes are fully dependent on id and the table is in 3NF

3.3. Data loading approach and record counts

I have handled data loading programmatically via a FastAPI and SQLModel on startup. When the application starts up, SQLModel.metadata.create_all(engine) creates the three tables if they do not exist. The function populate_if_empty(engine) checks whether the database is empty.

If empty, the application, the app fetches parks from the NPS API (/parks) and inserts them into park. Re-queries the persisted parks and builds a map from parkCode to park.id.

Fetches visitor centers from /visitorcenters and inserts them into visitorcenter, setting park_id by matching parkCode to the corresponding park.id.

For each park in the park's payload, it reads the embedded activities array and inserts up to 10 activities per park into activity, setting park_id to the current park's id and nps_id to the NPS activity ID.

Counting the number of records in each table

```
SELECT COUNT(*) FROM park;
SELECT COUNT(*) FROM visitorcenter;
SELECT COUNT(*) FROM activity;
```

3.4. Indexing strategy

```
CREATE INDEX idx_visitorcenter_park_id
   ON visitorcenter (park_id);

CREATE INDEX idx_activity_park_id
   ON activity (park_id);

CREATE INDEX idx_park_parkCode
   ON park (parkCode);

CREATE INDEX idx_activity_nps_id
   ON activity (nps_id);
```

4. SQL Query Demonstrations

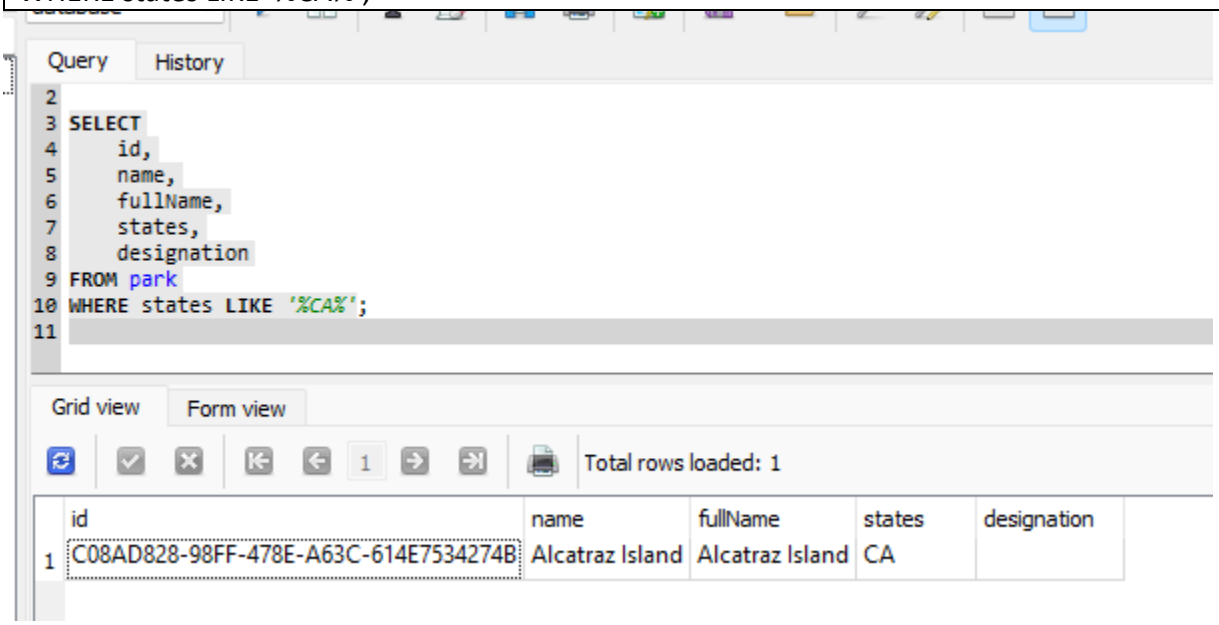
Q1. List all parks in a specific state (SELECT with WHERE)

Business question

Which national parks are located in a given U.S. state?

This query supports planning, regional analysis, and decision-making related to tourism management, resource allocation, and state-specific reporting.

```
SELECT
   id,
   name,
   fullName,
   states,
   designation
FROM park
WHERE states LIKE '%CA%';
```



Explanation

SELECT: Retrieves specific attributes (id, name, fullName, states, designation) instead of all columns, and this demonstrates projection.

WHERE clause: Applies a filter to restrict results to parks in a specific state.

Pattern matching with LIKE:

% is used to match any sequence of characters. This allows matching parks associated with multiple states stored in a single field.

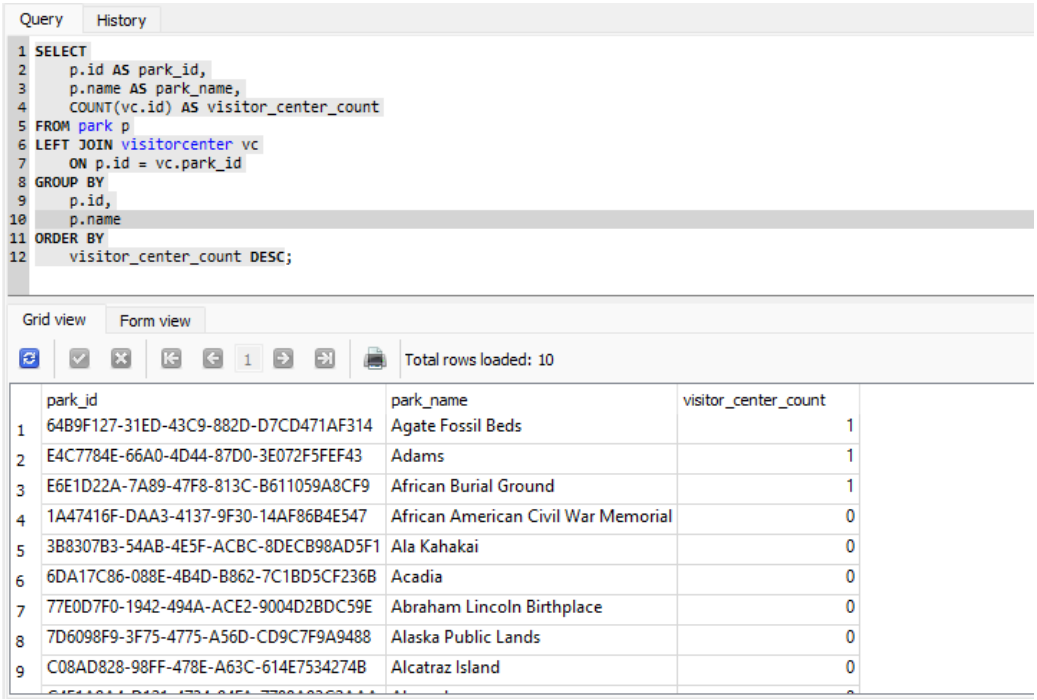
Q 2. Count visitor centers per park (JOIN + COUNT)

Business question

How many visitor centers does each national park have?

This query supports decisions related to visitor services, infrastructure planning, and staffing. Parks with a higher number of visitor centers may require greater operational resources, while parks with fewer or no visitor centers may indicate limited visitor access facilities.

```
SELECT
p.id AS park_id,
p.name AS park_name,
COUNT(vc.id) AS visitor_center_count
FROM park p
LEFT JOIN visitorcenter vc
ON p.id = vc.park_id
GROUP BY
p.id,
p.name
ORDER BY
visitor_center_count DESC;
```



Query History

```
SELECT
    p.id AS park_id,
    p.name AS park_name,
    COUNT(vc.id) AS visitor_center_count
FROM park p
LEFT JOIN visitorcenter vc
    ON p.id = vc.park_id
GROUP BY
    p.id,
    p.name
ORDER BY
    visitor_center_count DESC;
```

Grid view | Form view

Total rows loaded: 10

	park_id	park_name	visitor_center_count
1	64B9F127-31ED-43C9-882D-D7CD471AF314	Agate Fossil Beds	1
2	E4C7784E-66A0-4D44-87D0-3E072F5FEF43	Adams	1
3	E6E1D22A-7A89-47F8-813C-B611059A8CF9	African Burial Ground	1
4	1A47416F-DAA3-4137-9F30-14AF86B4E547	African American Civil War Memorial	0
5	3B8307B3-54AB-4E5F-ACBC-8DECB98AD5F1	Ala Kahakai	0
6	6DA17C86-088E-4B4D-B862-7C1BD5CF236B	Acadia	0
7	77E0D7F0-1942-494A-ACE2-9004D2BDC59E	Abraham Lincoln Birthplace	0
8	7D6098F9-3F75-4775-A56D-CD9C7F9A9488	Alaska Public Lands	0
9	C08AD828-98FF-478E-A63C-614E7534274B	Alcatraz Island	0

Explanation

Left join combines rows from park and visitorcenter based on the foreign key relationship.

The LEFT JOIN ensures that all parks appear in the results, even if no matching visitor center exists. **Aggregate function that is count, c**ounts the number of visitor centers associated with each park. **GROUP BY** groups rows by park so that the count is calculated per park. **ORDER BY** sorts the results in descending order of visitor center count, showing parks with the most visitor facilities first.

Q3: List activities offered at a specific park (2-table JOIN + WHERE)

Business question

What activities are available at a given national park?

This query supports visitor planning, tourism recommendations, and park-level service analysis. For example, park administrators and visitors may want to understand the recreational activities offered at a specific park.

```
SELECT
p.name AS park_name,
a.name AS activity_name
FROM park p
JOIN activity a
ON p.id = a.park_id
WHERE p.id = '77E0D7F0-1942-494A-ACE2-9004D2BDC59E'
ORDER BY a.name;
```

Query History

```
1 SELECT
2     p.name AS park_name,
3     a.name AS activity_name
4 FROM park p
5 JOIN activity a
6     ON p.id = a.park_id
7 WHERE p.id = '77E0D7F0-1942-494A-ACE2-9004D2BDC59E'
8 ORDER BY a.name;
9
```

Grid view Form view

Total rows loaded: 10

	park_name	activity_name
1	Abraham Lincoln Birthplace	Astronomy
2	Abraham Lincoln Birthplace	Birdwatching
3	Abraham Lincoln Birthplace	Food
4	Abraham Lincoln Birthplace	Guided Tours
5	Abraham Lincoln Birthplace	Hands-On
6	Abraham Lincoln Birthplace	Junior Ranger Program
7	Abraham Lincoln Birthplace	Picnicking
8	Abraham Lincoln Birthplace	Self-Guided Tours - Walking
9	Abraham Lincoln Birthplace	Stargazing
10	Abraham Lincoln Birthplace	Wildlife Watching

Explanation

INNER JOIN (2-table JOIN): Combines rows from the park table and the activity table using the foreign key relationship ensuring only activities that belong to the selected park are returned. WHERE clause: Filters the joined result set to a specific park. SELECT with aliases: Column aliases to improve readability and clarity of query results. ORDER BY: Sorts the activities alphabetically, making the output user-friendly and suitable for presentation.

Q4: Count activities per park (JOIN + GROUP BY)

Business question

How many activities are offered by each national park?

This query helps compare parks by the breadth of recreational opportunities available. It supports decisions around visitor experience planning, benchmarking parks, and identifying parks with extensive activity offerings.

```
SELECT
p.id AS park_id,
p.name AS park_name,
COUNT(a.id) AS activity_count
FROM park p
LEFT JOIN activity a
ON p.id = a.park_id
GROUP BY
p.id,
p.name
ORDER BY
activity_count DESC;
```

Query History

```
1  SELECT
2      p.id AS park_id,
3      p.name AS park_name,
4      COUNT(a.id) AS activity_count
5  FROM park p
6  LEFT JOIN activity a
7      ON p.id = a.park_id
8  GROUP BY
9      p.id,
10     p.name
11 ORDER BY
12     activity_count DESC;
13
```

Grid view Form view

Total rows loaded: 10

	park_id	park_name	activity_count
1	3B8307B3-54AB-4E5F-ACBC-8DECB98AD5F1	Ala Kahakai	10
2	64B9F127-31ED-43C9-882D-D7CD471AF314	Agate Fossil Beds	10
3	6DA17C86-088E-4B4D-B862-7C1BD5CF236B	Acadia	10
4	77E0D7F0-1942-494A-ACE2-9004D2BDC59E	Abraham Lincoln Birthplace	10
5	C4E1A9A4-D121-4734-94FA-7788A93C2AAA	Alagnak	10
6	E4C7784E-66A0-4D44-87D0-3E072F5FEF43	Adams	8
7	E6E1D22A-7A89-47F8-813C-B611059A8CF9	African Burial Ground	7
8	C08AD828-98FF-478E-A63C-614E7534274B	Alcatraz Island	6
9	1A47416F-DAA3-4137-9F30-14AF86B4E547	African American Civil War ...	2
10	7D6098F9-3F75-4775-A56D-CD9C7F9A9488	Alaska Public Lands	0

Explanation

The LEFT JOIN is used to confirm that the parks are managed even in when there are no activities going on. This will preserve parks that do not have recreation data. The GROUP BY is used in calculating activities that belongs to certain park regardless of whether it has recreation data or not. ORDER helps in ranking all the parks from the one with highest number of activities to the one with the least.

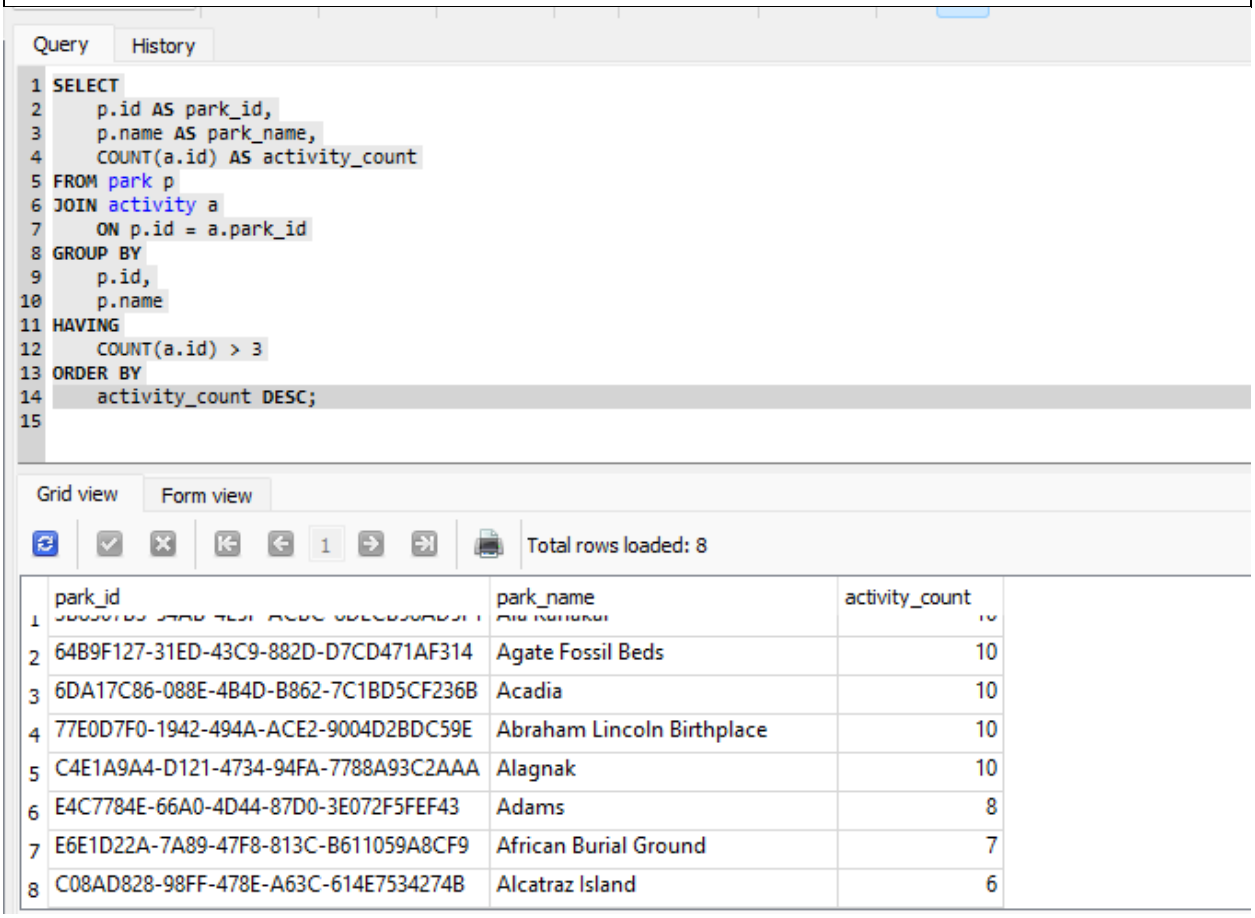
Q5: Find parks with more than 3 activities (GROUP BY + HAVING)

Business question

Which national parks offer a wide range of activities (more than three)?

This query supports comparative analysis of parks based on activity diversity. It helps identify parks with richer recreational offerings, which can inform visitor recommendations, marketing priorities, and resource allocation decisions.

```
SELECT
p.id AS park_id,
p.name AS park_name,
COUNT(a.id) AS activity_count
FROM park p
JOIN activity a
ON p.id = a.park_id
GROUP BY
p.id,
p.name
HAVING
COUNT(a.id) > 3
ORDER BY
activity_count DESC;
```



Query History

```
1  SELECT
2      p.id AS park_id,
3      p.name AS park_name,
4      COUNT(a.id) AS activity_count
5  FROM park p
6  JOIN activity a
7      ON p.id = a.park_id
8  GROUP BY
9      p.id,
10     p.name
11 HAVING
12     COUNT(a.id) > 3
13 ORDER BY
14     activity_count DESC;
15
```

Grid view Form view

Total rows loaded: 8

	park_id	park_name	activity_count
2	64B9F127-31ED-43C9-882D-D7CD471AF314	Agate Fossil Beds	10
3	6DA17C86-088E-4B4D-B862-7C1BD5CF236B	Acadia	10
4	77E0D7F0-1942-494A-ACE2-9004D2BDC59E	Abraham Lincoln Birthplace	10
5	C4E1A9A4-D121-4734-94FA-7788A93C2AAA	Alagnak	10
6	E4C7784E-66A0-4D44-87D0-3E072F5FEF43	Adams	8
7	E6E1D22A-7A89-47F8-813C-B611059A8CF9	African Burial Ground	7
8	C08AD828-98FF-478E-A63C-614E7534274B	Alcatraz Island	6

Explanation

After grouping the parks, HAVING is used in appraising the parks which holds more activities than expected. Afterwards, ORDER clause aids in arranging all selected parks based on the number of activities. This indicates that the park having more activities appears first.

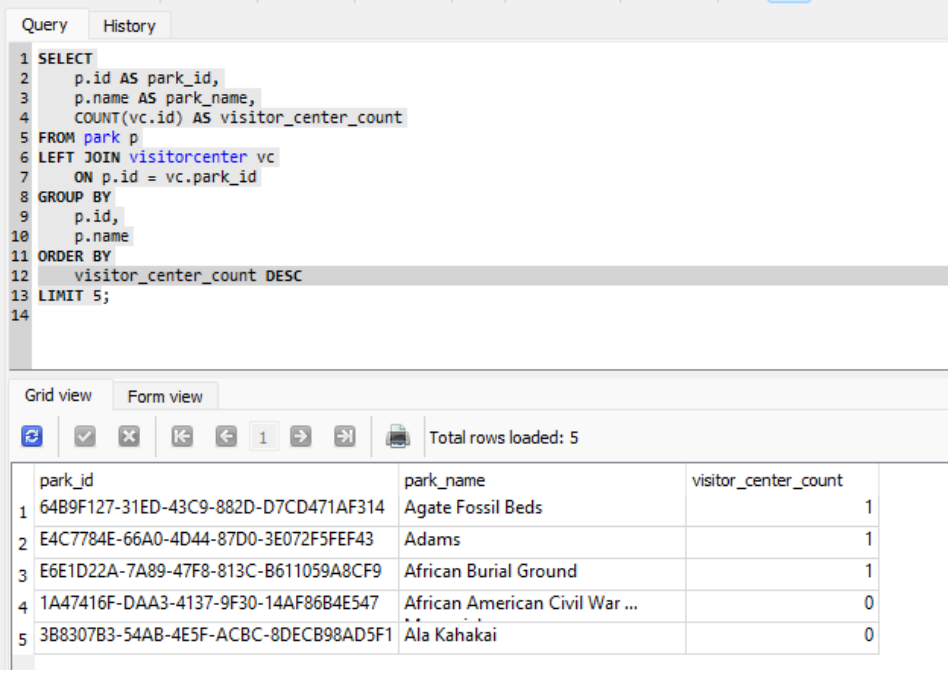
Q6: List top 5 parks by number of visitor centers (ORDER BY + LIMIT)

Business question

Which national parks have the highest number of visitor centers?

This query helps identify parks with the largest visitor service infrastructure. Such information supports decisions related to staffing levels, budget allocation, and prioritization of visitor information services.

```
SELECT
p.id AS park_id,
p.name AS park_name,
COUNT(vc.id) AS visitor_center_count
FROM park p
LEFT JOIN visitorcenter vc
ON p.id = vc.park_id
GROUP BY
p.id,
p.name
ORDER BY
visitor_center_count DESC
LIMIT 5;
```



Query History

```
SELECT
    p.id AS park_id,
    p.name AS park_name,
    COUNT(vc.id) AS visitor_center_count
FROM park p
LEFT JOIN visitorcenter vc
    ON p.id = vc.park_id
GROUP BY
    p.id,
    p.name
ORDER BY
    visitor_center_count DESC
LIMIT 5;
```

Grid view Form view Total rows loaded: 5

	park_id	park_name	visitor_center_count
1	64B9F127-31ED-43C9-882D-D7CD471AF314	Agate Fossil Beds	1
2	E4C7784E-66A0-4D44-87D0-3E072F5FEF43	Adams	1
3	E6E1D22A-7A89-47F8-813C-B611059A8CF9	African Burial Ground	1
4	1A47416F-DAA3-4137-9F30-14AF86B4E547	African American Civil War ...	0
5	3B8307B3-54AB-4E5F-ACBC-8DECB98AD5F1	Ala Kahakai	0

Explanation

ORDER BY categorizes the parks based on the number of visitor center from the one with least to the one with highest number. Through this, the managers can identify which parks have Largest Infrastructure Footprint. LIMIT 5 will be used in the analysis where the managers need to access the top five parks in the region, meaning that other parks will not be considered.

Q7: Find the most common activity across parks (Subquery + WHERE)

Business question

Which activity is offered by the largest number of parks?

This query helps identify the most common recreational activity across parks. Such information can support national-level planning, marketing strategies, and prioritization of facilities that serve the most popular activities.

```
SELECT
name,
COUNT(DISTINCT park_id) AS park_count
FROM activity
GROUP BY
name
HAVING
COUNT(DISTINCT park_id) = (
SELECT
```

```
MAX(activity_count)
FROM (
SELECT
COUNT(DISTINCT park_id) AS activity_count
FROM activity
GROUP BY name
)
);
```

Query History

```
SELECT
    name,
    COUNT(DISTINCT park_id) AS park_count
FROM activity
GROUP BY
    name
HAVING
    COUNT(DISTINCT park_id) = (
        SELECT
            MAX(activity_count)
        FROM (
            SELECT
                COUNT(DISTINCT park_id) AS activity_count
            FROM activity
            GROUP BY name
        )
    );
```

Grid view Form view

Total rows loaded: 1

	name	park_count
1	Guided Tours	5

Explanation

The inner QUERY determines the number of parks which provides a certain activity to the visitor. The Query on the outer side apprehends activities whose number is the same as totals required in subquery. Therefore, this aspect only enlightens park activity with largest availability at national level.

Q8: List visitor centers and their park names (JOIN + SELECT)

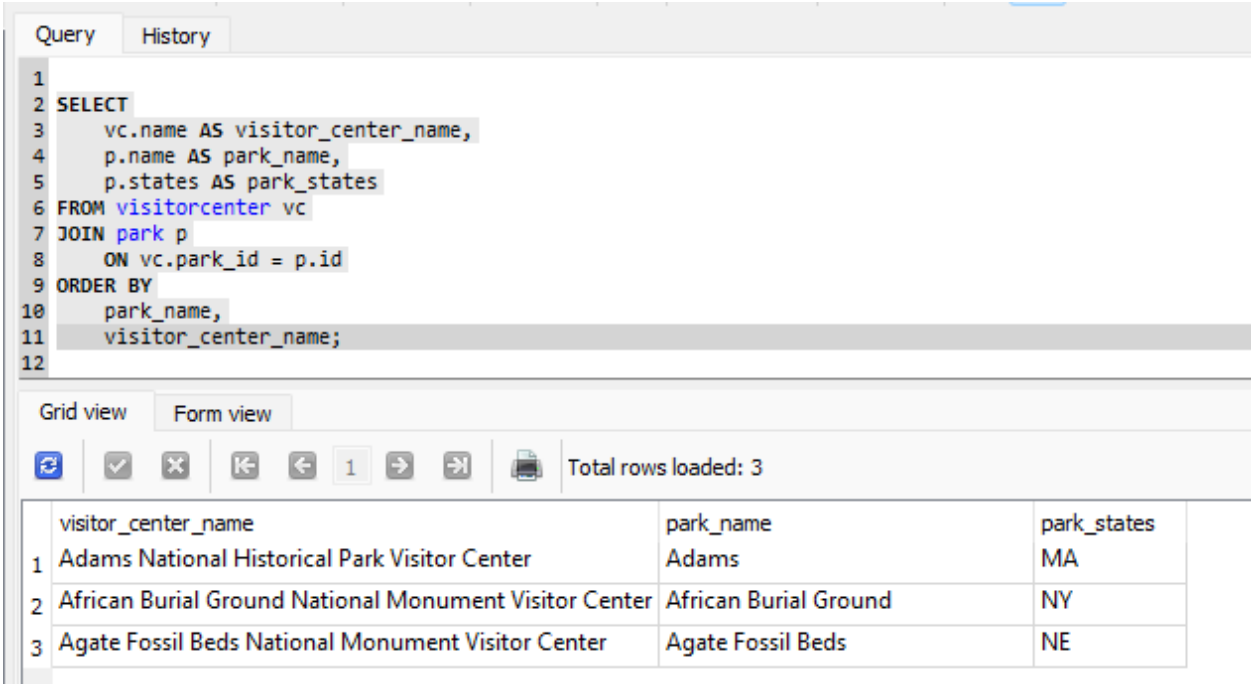
Business question

Which visitor centers belong to which national parks?

This query provides a clear mapping between visitor centers and their associated parks. It supports operational planning, visitor information services, and reporting needs, such as displaying visitor centers together with the parks they serve.

```
SELECT
```

```
vc.name AS visitor_center_name,
p.name AS park_name,
p.states AS park_states
FROM visitorcenter vc
JOIN park p
ON vc.park_id = p.id
ORDER BY
park_name,
visitor_center_name;
```



Query History

```
1
2  SELECT
3      vc.name AS visitor_center_name,
4      p.name AS park_name,
5      p.states AS park_states
6  FROM visitorcenter vc
7  JOIN park p
8      ON vc.park_id = p.id
9  ORDER BY
10     park_name,
11     visitor_center_name;
12
```

Grid view | Form view

Total rows loaded: 3

	visitor_center_name	park_name	park_states
1	Adams National Historical Park Visitor Center	Adams	MA
2	African Burial Ground National Monument Visitor Center	African Burial Ground	NY
3	Agate Fossil Beds National Monument Visitor Center	Agate Fossil Beds	NE

Explanation

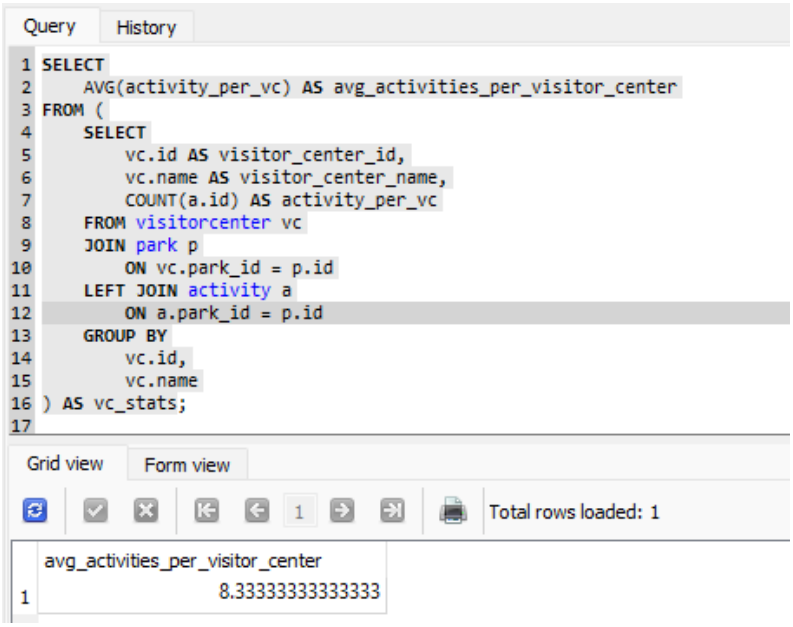
The JOIN will match every visitor to specific park utilizing Foreign Key Relationship. When we select both park and visitor center, we obtain a clear information that can be used in UI displays and in reference tables. Finally, ORDER BY is incorporated to sort visitor center and park name to ensure efficient navigability.

Q9: Calculate average number of activities per visitor center (Aggregation + JOIN)

Business question

On average, how many activities are available to visitors at each visitor center?
This query provides a high-level measure of how “rich” the visitor experience is across all visitor centers. It supports strategic decisions about whether some visitor centers may need more facilities or promotion to match the activity offerings available at others.

```
SELECT
AVG(activity_per_vc) AS avg_activities_per_visitor_center
FROM (
SELECT
vc.id AS visitor_center_id,
vc.name AS visitor_center_name,
COUNT(a.id) AS activity_per_vc
FROM visitorcenter vc
JOIN park p
ON vc.park_id = p.id
LEFT JOIN activity a
ON a.park_id = p.id
GROUP BY
vc.id,
vc.name
) AS vc_stats;
```



Explanation

Inner query highlights activities available in every visitor center from the parent park. The results on the outer query therefore allows the AVG to calculate system average. This results to a two-level aggregation that is important in complex analytics, meaning that they cannot be embraced for single GROUP.

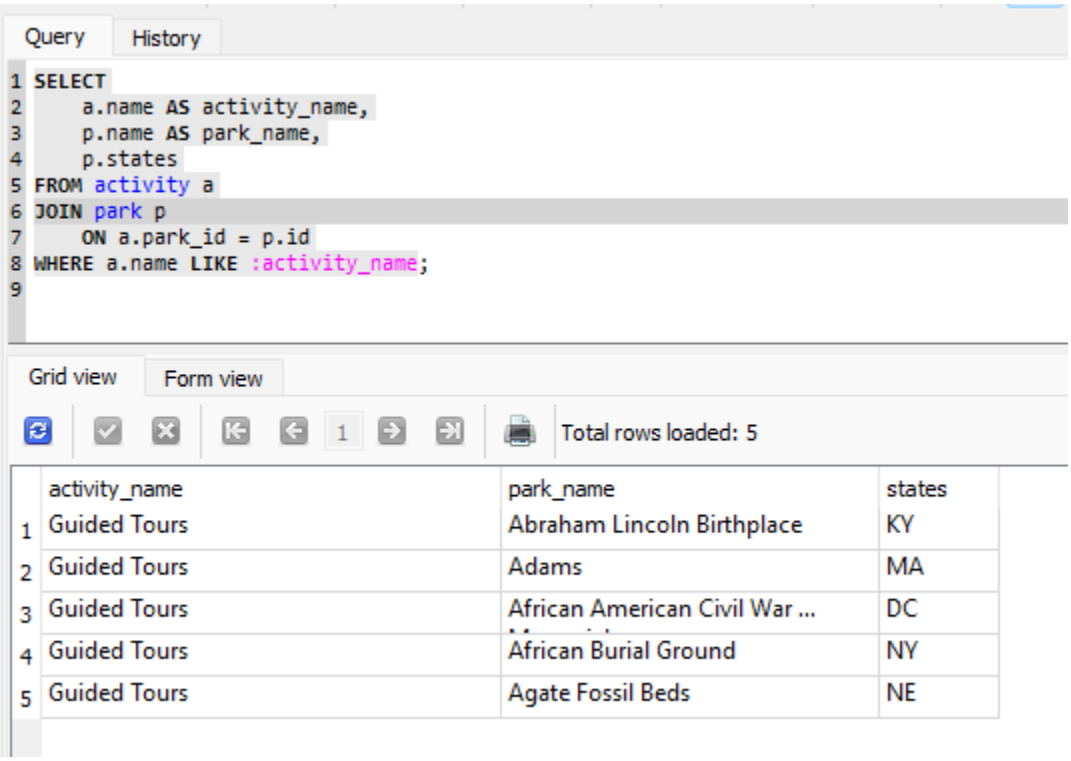
Q10: Parameterized activity search by name (Parameterized WHERE clause)

Business question

Which parks offer a specific activity searched by the user?

This query supports interactive search functionality, allowing users to look up activities of interest and identify the parks where those activities are available.

```
SELECT
a.name AS activity_name,
p.name AS park_name,
p.states
FROM activity a
JOIN park p
ON a.park_id = p.id
WHERE a.name LIKE :activity_name;
```



	activity_name	park_name	states
1	Guided Tours	Abraham Lincoln Birthplace	KY
2	Guided Tours	Adams	MA
3	Guided Tours	African American Civil War ...	DC
4	Guided Tours	African Burial Ground	NY
5	Guided Tours	Agate Fossil Beds	NE

Explanation

Parameterized WHERE clause permits filtering that uses user-provided activity to ensure security and flexibility through prevention of SQL injection. At the same time, JOIN confirms that every user can identify which park offers certain activity. The user will have an opportunity to decide which center to visit at a given time because the desired activity is offered there.

5. Backend API Implementation

5.1. FastAPI Structure and Organization

The backend API is implemented using FastAPI and follows a modular and maintainable structure.

main.py

Serves as the application entry point. It initializes the FastAPI instance, configures middleware, creates database tables on startup, and registers all routers.

models.py

Contains SQLModel classes (Park, VisitorCenter, Activity) that define both the database schema and the response serialization structure.

database.py

Centralizes the SQLite engine configuration and allows consistent session management across the application.

/routers

Each resource type has its own router module (parks.py, visitor_centers.py, activities.py). This approach improves readability and scalability by grouping related endpoints together.

5.2. Endpoint Design and RESTful Principles

The API follows RESTful design principles, using clear and consistent URL patterns and HTTP methods

/parks, /visitor_centers, /activities represent collections.

/parks/{park_id}, /visitor_centers/{visitor_center_id}, /activities/{activity_id} represent individual resources.

HTTP methods
The current implementation focuses on GET operations for data retrieval, ensuring safe access to resources.

Consistent response formats

All endpoints return JSON-formatted responses

5.3. Error Handling Approach

The API implements explicit and meaningful error handling to ensure robustness and usability

HTTPException

FastAPI's HTTPException is used consistently to return appropriate HTTP status codes and human-readable error messages.

5.4. Data Validation with Pydantic

Data validation is handled using Pydantic

Path parameter validation

Path variables such as park_id, visitor_center_id, and activity_id are validated using constraints

Minimum and maximum string length for IDs

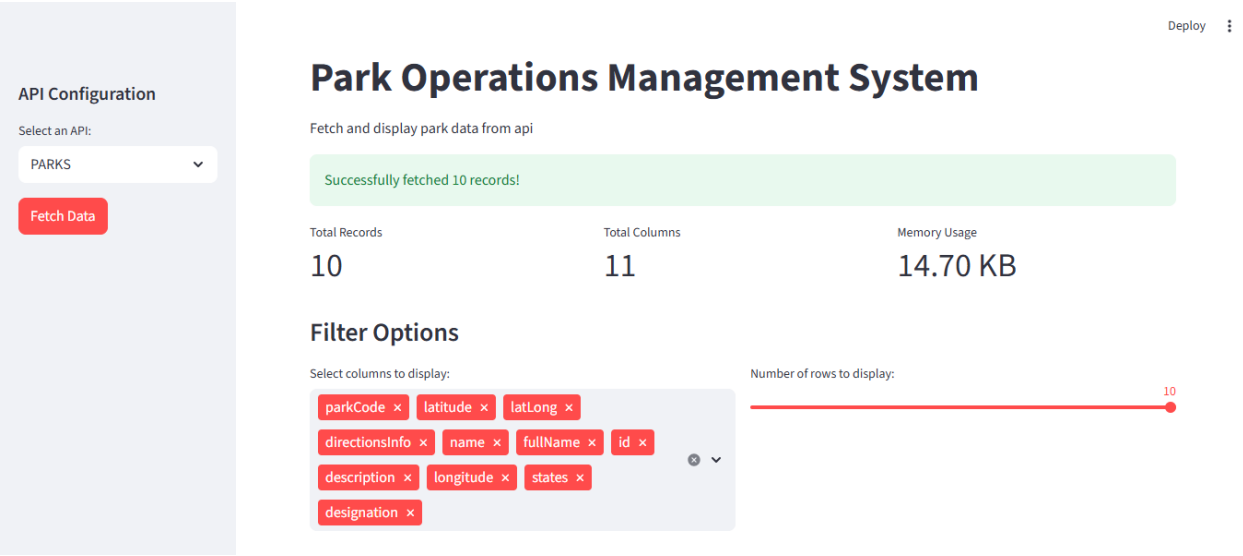
Numeric bounds for integer identifiers

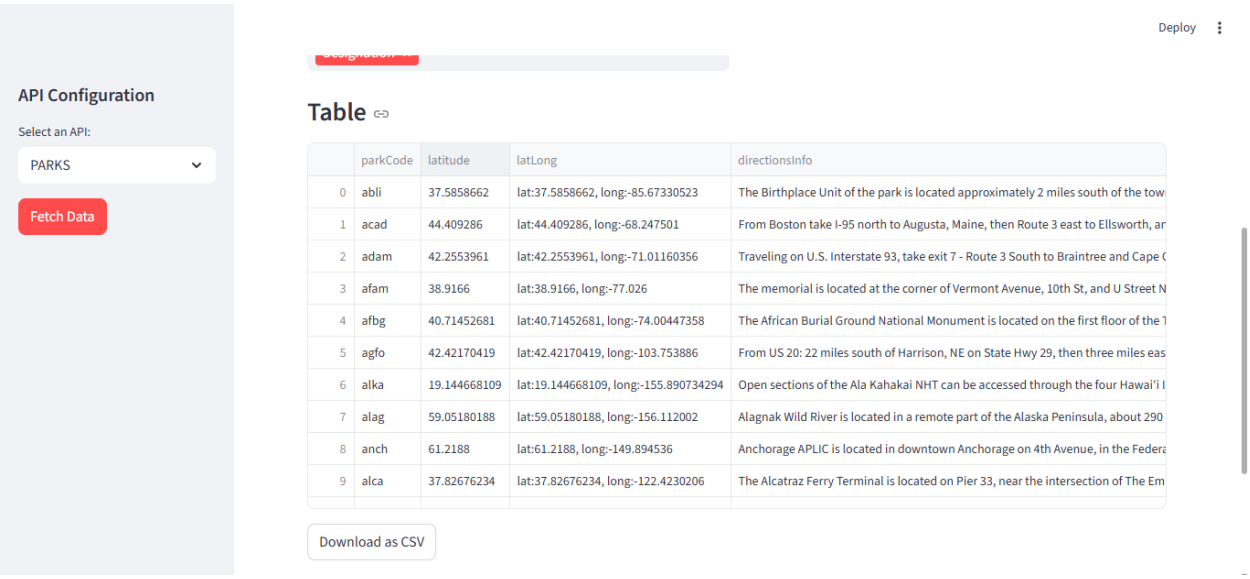
Frontend Implementation

I implemented the frontend using streamlit. The main reason for choosing streamlit is because it is easy to use, integrates well with APIs and it is faster. The interface provided will therefore allow effective display of data that do not need complex framework. I consumed parks, activities and visitor centers endpoints. I displayed the data in tables which are filterable and searchable.

The main features of frontend comprise;

1. Data retrieval: the data concerning the park will be retrieved and finally converted to a Data Frame to ensure easy calculation and manipulation.
2. Visualization; the frontend contains tables that displays data concerning the park activities.
3. User interface: the frontend contains sections where the user can search for certain park locations, activities offered, and visitor centers. At this point, the visitor does not need to consult with the staff members when they want to know the activities available; instead, they have to browse and get an idea about the park.





Setup and Usage Instructions

- From the base folder, cd to the backend folder
- Open the command prompt while in the same folder and run the following command
- ‘python3 -m venv venv’ to create a virtual environment named venv
- run this command to activate the venv ‘venv\Scripts\activate’
- Installed the dependencies ‘pip install -r requirements.txt’
- Run ‘uvicorn main:app --reload --host 0.0.0.0 --port 8000’
- The backend will run and listen on port 8000

The frontend does not need to have a virtual environment.

Run the following command to install the required packages

‘pip install streamlit pandas requests’

This should be run while inside the frontend folder

Run ‘streamlit run app.py’

The frontend page will be opened on the browser. Select an option that is PARK,CENTERS or ACTIVITIES and then click Fetch Data button.

Testing and Challenges

I tested the API endpoints using postman. This was to guarantee I have verified every API independently based on park's performance. After verification, the Parameterized Routes appropriately responded to invalid and valid IDs. I also evaluated how API will be used in handling missing information about the park and other non-existent data. Finally, I confirmed all queries presented in the table behaved suitably when loading certain data.

I encountered errors based on model mappings

1. There were some errors that befell because the SQLite schema and SQLModel did not match as they needed appropriate configuration of attribute types and names.
2. There was complexity in data integration because the API structure required me to include complex logic.

Although I experienced all these challenges, the implemented system will ensure that data about the park activities is well managed and therefore can be used in the decision-making process. The managers will rely on this factual data rather than manual spreadsheets when planning for the park's activities.

GITHUB REPOSITORY

URL: **https://github.com/hamadal00/FinalProject.git**