IEE 305 Term Project Proposal

Due Date: November 25th, 11:59 PM **Points:** 50

1. Group Information (3 points)

Project Title: NPS Visitor Demand Planner: Regional Capacity & Staffing Support Tool

Selected API Option: National Parks System

Group Members:

Name	ASU Email	Primary Responsibilities
Esteban Dozal	edozal3@asu.edu	Everything

2. Industrial Engineering Problem Statement (10 points)

Problem Context (1-2 paragraphs)

Operations managers in the National Park Service (NPS) must plan staffing, maintenance, and visitor services under highly seasonal and uneven visitor demand. Some parks experience extreme peaks in summer and holidays, while others have more evenly distributed visitation. At the same time, regional offices need to understand how total demand is distributed across parks in different regions to allocate limited staff, budget, and infrastructure investments efficiently.

Today, planners typically download separate spreadsheets or PDF reports from the NPS Visitor Use Statistics system and manually filter and summarize data for individual parks. This process is time-consuming, error-prone, and makes it difficult to compare multiple parks or regions over time. It also limits the ability to quickly answer operational questions such as “Which parks in this region are approaching capacity during peak months?” or “Which parks show sustained demand growth and may require additional infrastructure?” Location-based analysis (e.g., mapping high-demand parks) is also difficult without a structured spatial database.

While the National Park Service provides a public Visitor Use Statistics Explorer, the tool is primarily designed for simple visualization of historical data and does not support the analytical needs of operations managers or industrial engineers. It lacks a relational database structure, does not allow multi-park or multi-region comparative analysis, and cannot compute derived operational metrics such as seasonal variability, peak-to-average ratios, growth trends, or regional demand distribution. It also does not integrate location data for spatial analysis and cannot be used programmatically for decision-support tasks.

In contrast, the proposed system consolidates official NPS metadata and visitation statistics into a normalized database, enabling advanced SQL queries, cross-park analytics, capacity planning, and data-driven decision-making. The system is designed specifically to support resource allocation, staffing optimization, maintenance scheduling, and long-term planning, core responsibilities of NPS operations managers and Industrial Engineering professionals.

Solution Approach (1 paragraph)

Our system, the NPS Visitor Demand Planner, consolidates NPS park metadata and monthly visitor statistics into a relational database, exposing analytical queries through a FastAPI backend and a Streamlit frontend. Park metadata, including names, state, designation, and coordinates (latitude/longitude), is fetched from the NPS public API. Monthly visitor counts for recent years (e.g., 2022–2024) are loaded from official NPS visitor statistics CSV reports. The system enables engineers to explore seasonal and long-term visitation patterns by park and by region, compute rankings (e.g., the top 10 busiest parks in a year), and identify parks with unusually high or rapidly growing demand. These capabilities support data-driven decisions for staffing, maintenance scheduling, capacity planning, and potential infrastructure changes.

Target Users and Use Cases

Target Users: NPS regional operations managers, Park superintendents and planners, Industrial Engineers supporting visitor flow and capacity planning

Primary Use Cases:

1. Identify peak and off-peak months for a given park to align staffing levels and maintenance windows with visitor demand.

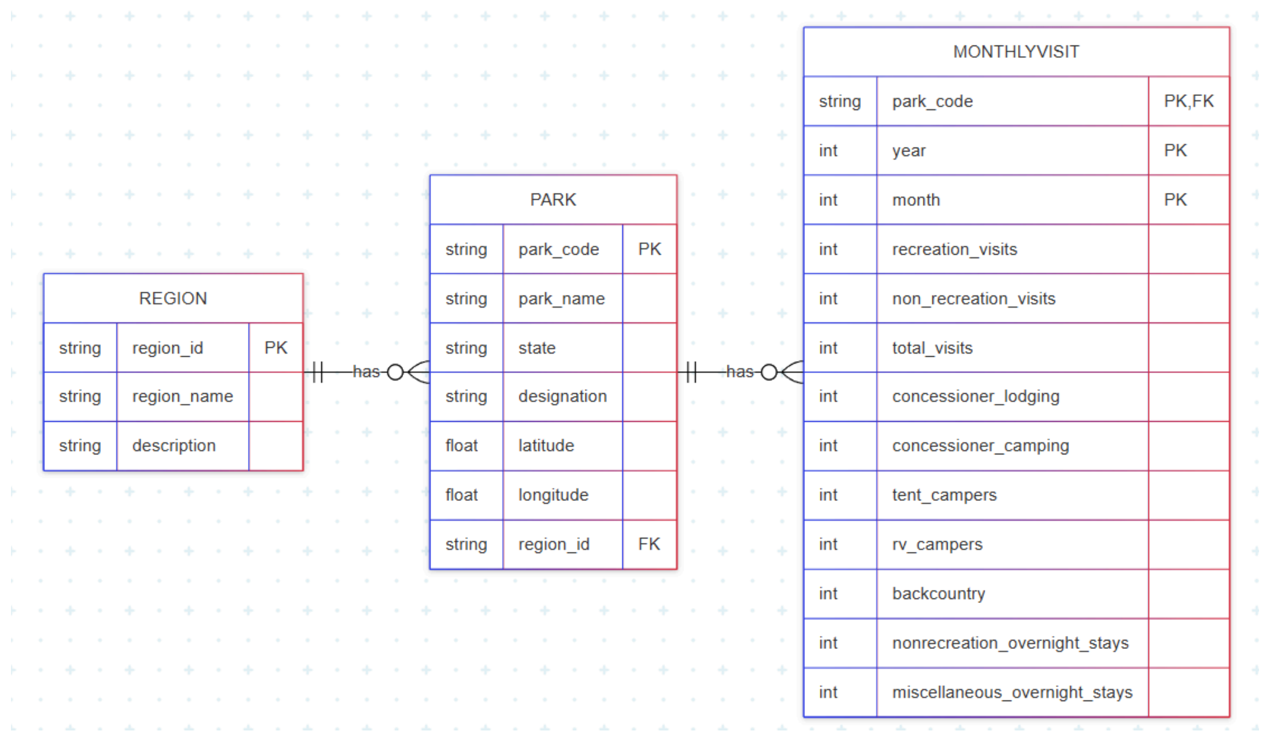
2. Compare total annual visitation across parks in the same region to prioritize resource allocation (budget, staff, infrastructure projects).

3. Detect parks whose annual visitation exceeds regional or system-wide averages or shows high growth, indicating potential congestion and capacity issues that may require interventions such as shuttle services, parking expansions, or reservation systems.

---

3. Entity-Relationship Model (10 points)

ER Diagram



Entity Descriptions

Entity 1: Region - **Description:** Represents an NPS administrative region used to aggregate parks and analyze demand at a regional level.

- **Attributes:**

- `region_id` (PRIMARY KEY) - [Short code for the region, e.g., IMR, TEXT]

- `region_name` - [Region name e.g. Intermountain Region, TEXT]

- `description` - [Optional text description of the region, TEXT, (nullable)]

Entity 2: PARK - **Description:** Represents an individual NPS park unit that receives visitors (e.g., Grand Canyon NP).

- **Attributes:**

- `park_code` (PRIMARY KEY) - [Official NPS park code e.g., grca, TEXT]

- name – [Full park name (e.g., "Grand Canyon National Park"), TEXT]

- state – [Two-letter state or list of state abbreviations (e.g., "AZ"), TEXT]

- designation – [Park designation (e.g., "National Park"), TEXT]

- region_id – [Foreign key referencing Region.region_id, TEXT (NOT NULL)]

- latitude – [Park latitude in decimal degrees, REAL (nullable)]

- longitude – [Park longitude in decimal degrees, REAL (nullable)]

Entity 3: MonthlyVisit - **Description: Represents monthly visitor use statistics for a given park, year, and month.**

- **Attributes:**

- park_code – (PRIMARY KEY) [Foreign key referencing Park.park_code, TEXT (NOT NULL)]

- year – (PRIMARY KEY) [Calendar year (e.g., 2023), INTEGER (NOT NULL)]

- month – (PRIMARY KEY) [Calendar month number (1–12), INTEGER (NOT NULL)]

- recreation_visits – [Number of recreation visits in that month, INTEGER]

- non_recreation_visits – [Number of non-recreation visits, INTEGER]

- total_visits – [Total visits (recreation + non-recreation), INTEGER]

- concessioner_lodging - [Number of concessioner lodgings in that month, INTEGER]

- concessioner_camping - [Number of concessioner camping in that month, INTEGER]

- tent_campers - [Number of tent camper visits in that month, INTEGER]

-rv_campers - [Number of RV camper visits in that month, INTEGER]

- backcountry - [Number of backcountry visits in that month, INTEGER]

- nonrecreation_overnight_stays - [Number of nonrecreation overnight stays in that month, INTEGER]

- miscellaneous_overnight_stays - [Number of miscellaneous overnight stays in that month, INTEGER]

Relationship Descriptions

Relationship 1: Region ↔ **Park** - **Type:** [One Region has Many Parks (1:N)] - **Description:** Each park belongs to exactly one NPS administrative region; each region supervises multiple parks.

Relationship 2: Park ↔ **MonthlyVisit** - **Type:** One Park has Many MonthlyVisit records (1:N)

Description: Each park has visitor use statistics for multiple months and years; each MonthlyVisit record represents one park's statistics for a specific year and month.

---

4. Relational Data Model (10 points)

Table Schemas

```sql
CREATE TABLE region (
    region_id TEXT PRIMARY KEY,
    region_name TEXT NOT NULL,
    description TEXT
);

CREATE TABLE park (
    park_code TEXT PRIMARY KEY,
    park_name TEXT NOT NULL,
    state TEXT NOT NULL,
    designation TEXT NOT NULL,
    region_id TEXT NOT NULL,
    latitude REAL,
    longitude REAL,
    FOREIGN KEY (region_id) REFERENCES region(region_id)
);

CREATE TABLE  monthly_visit (
    park_code TEXT NOT NULL,
    year INTEGER NOT NULL,
    month INTEGER NOT NULL,
    recreation_visits INTEGER,
    non_recreation_visits INTEGER,
    total_visits INTEGER,
    concessioner_lodging INTEGER,
    concessioner_camping INTEGER,
    tent_campers INTEGER,
    rv_campers INTEGER,
    backcountry INTEGER,
    nonrecreation_overnight_stays INTEGER,
    miscellaneous_overnight_stays INTEGER,
    PRIMARY KEY (park_code, year, month),
    FOREIGN KEY (park_code) REFERENCES park(park_code)
);
```

Normalization Analysis (3NF)

First Normal Form (1NF): - ✅ All attributes contain atomic values - **Justification:** [Explain how your tables meet 1NF]

Second Normal Form (2NF): - ✅ No partial dependencies - **Functional Dependencies:** - Table 1: [e.g., "earthquake_id → magnitude, depth, location"] - Table 2: [List dependencies] - Table 3: [List dependencies]

Third Normal Form (3NF): - ✅ No transitive dependencies - **Justification:** [Explain how your tables meet 3NF]

---

5. Sample Scraped Data (5 points)

Data Fetching Status

- **API Endpoint(s) Used:**
  NPS Developer API (parks metadata): https://developer.nps.gov/api/v1/parks
  474 Records
  Visitor Statistics Source:
  NPS Visitor Use Statistics Data Portal / Statistical Abstract:
  Monthly park-level visitation CSV/Excel reports (e.g., 2022–2024)
  14172 Records
- **Data Transformation:**
  - From the CSV:
  - Extract distinct values from the `Region` column and map each to a `region_id` and `region_name`.
  - Use `UnitCode` as the park_code and the CSV `Region` value (via mapping) to assign `region_id` for each park.
  - Use `Year`, `Month`, and the various visit count columns to populate the `monthly_visit` table.

  From the NPS Parks API:
  For each park_code present in the CSV, call the parks API and enrich the `park` records with metadata:
  parkCode → park.park_code
  fullName → park.park_name
  states → park.state
  designation → park.designation
  latLong → parsed into park.latitude and park.longitude (decimal degrees)

Sample Data (3+ records per table)

Table 1: `region`

region_id	region_name	description
AKR	Alaska Region	Parks located in Alaska
IMR	Intermountain Region	Parks in the Intermountain Region
MWR	Midwest Region	Parks in the central United States
NCR	National Capital Region	Parks in and around Washington, DC
NER	Northeast Region	Parks in the Northeastern United States
PWR	Pacific West Region	Parks in the Western and Pacific United States
SER	Southeast Region	Parks in the Southeastern United States and Caribbean

Table 2: `park`

park_code	park_name	state	designation	region_id	latitude	longitude
grca	Grand Canyon National Park	AZ	National Park	IMR	36.0001165336	-112.121516363
zion	Zion National Park	UT	National Park	IMR	37.29839254	-113.0265138
yose	Yosemite National Park	CA	National Park	PWR	37.84883288	-119.5571873

Table 3: `[table_name]`

park_code	year	month	recreation_visits	non_recreation_visits	total_visits	concessioner_lodging
grca	2023	7	750000	20000	770000	30000
zion	2023	7	650000	15000	665000	25000
yose	2023	7	650000	10000	660000	28000

More columns will be included in the actual table*

---

6. SQL Query Specifications (7 points)

[For each query, specify the business question and SQL concepts. Implementation comes later.]

Your 10 queries must collectively cover these 7 SQL concepts: 1. Basic Filtering (SELECT with WHERE) 2. JOIN Operations (2+ tables, including at least one 3-table join) 3. Aggregation Functions (COUNT, SUM, AVG, MIN, or MAX) 4. GROUP BY with HAVING 5.

Subqueries or CTEs 6. ORDER BY with LIMIT 7. Parameterized Query (SQL injection prevention)

Note: Some queries can demonstrate multiple concepts simultaneously.

Query 1:

- Business Question: For a given park and year, what are the monthly total visits, and which months exceed a chosen demand threshold (e.g., 500,000 visits)?

- SQL Concepts Demonstrated: Basic Filtering (WHERE), ORDER BY

Query 2:

- Business Question: List all parks with their region names and total annual visits in a selected year.

- SQL Concepts Demonstrated: JOIN Operations (region–park–monthly_visit), Aggregation Functions (SUM), GROUP BY

Query 3:

- Business Question: What is the average monthly total visitation for each park over a selected multi-year period (e.g., 2022–2024)?

- SQL Concepts Demonstrated: Aggregation Functions (AVG), GROUP BY, Basic Filtering (WHERE year BETWEEN ...)

Query 4:

- Business Question: Which parks have an average peak-season (June–August) monthly visitation above a specified threshold (e.g., 600,000 visits)?

- SQL Concepts Demonstrated: Aggregation Functions, GROUP BY with HAVING, Basic Filtering (WHERE month IN (6,7,8))

Query 5:

- Business Question: Which parks have total annual visits greater than the system-wide average annual visits in the same year?

- SQL Concepts Demonstrated: Subquery or CTE to compute system-wide average, Aggregation Functions, JOIN Operations, GROUP BY

Query 6:

- Business Question: What are the top 10 most visited parks in a given year based on total annual visits?

- SQL Concepts Demonstrated: Aggregation Functions (SUM), GROUP BY, ORDER BY with LIMIT

Query 7:

- Business Question: For each region, what is the total annual visitation (sum across all parks) in a selected year, and how do regions rank from highest to lowest?

- SQL Concepts Demonstrated: 3-table JOIN (region–park–monthly_visit), Aggregation Functions, GROUP BY, ORDER BY

Query 8:

- Business Question: For a selected park, what is the month-to-month change in total visits within a year (to help identify sudden spikes or drops in demand)?

- SQL Concepts Demonstrated: Subquery or self-JOIN on monthly_visit, Basic Filtering, ORDER BY

Query 9:

- Business Question: For a given region and time window (e.g., 2022–2024), which parks show the highest percentage growth in total annual visitation?

- SQL Concepts Demonstrated: JOIN Operations, Aggregation Functions, Subquery/CTE for growth calculation, GROUP BY, ORDER BY

Query 10:

- Business Question: Given a region_id, year, and optional minimum total_visits parameter from the frontend, return all parks in that region with their annual total visits that meet or exceed the threshold (used as a parameterized query in the API).

- SQL Concepts Demonstrated: Parameterized Query (prevent SQL injection), Basic Filtering, JOIN Operations, Aggregation Functions, GROUP BY

---

7. API Integration Plan (2 points)

Primary API Endpoint(s): NPS Parks Metadata: - https://developer.nps.gov/api/v1/parks

Authentication: Obtained API key by signing up through the NPS website and was given through email.

Data Fetching Strategy:

- One-time initial population: Use a Python script (`fetch_data.py`) to call the NPS parks endpoint, parse the JSON response, and populate the `park` table.

- Visitor statistics loading:

  - Download monthly visitor statistics CSV files (e.g., 2022–2024) from the NPS Visitor Use Statistics portal.

  - Use the same script or a separate ETL script to transform the CSV rows into `monthly_visit` records using (park_code, year, month, and visit counts).

- Runtime behavior:

  - After the database is populated, all analytical queries are executed against the local SQLite database via the FastAPI backend.

  - The NPS API is used primarily for initial data acquisition and periodic updates, not for each end-user query.

Expected Record Counts: - Table 1: [7 records] - Table 2: [~474 records] - Table 3: [~14172 records]

Data Mapping:

API Field	Database Table	Database Column	Transformation
API.parks[].parkCode	park	park_code	Direct copy; used to join with CSV.UnitCode
API.parks[].fullName	park	park_name	Direct copy
API.parks[].states	park	state	Direct copy (e.g., "AZ" or "AZ,UT")
API.parks[].designation	park	designation	Direct copy (e.g., "National Park")
API.parks[].latitude	park	latitude	Convert string to REAL
API.parks[].longitude	park	longitude	Convert string to REAL

---

8. Technology Stack (3 points)

Backend: - Database Access: `SQLModel`

(SQLModel provides ORM-style models that also serve as Pydantic schemas, making it easy to define database tables and API responses in one place.)

 - Other Libraries: requests (for calling the NPS Parks API)

  - pandas (for loading and transforming the visitor statistics CSV files)

  - python-dotenv (to store the NPS API key securely)

Frontend: - Framework: Streamlit - Visualization: Plotly, Matplotlib (for Streamlit), potentially other integration for map visualizations

Justification: SQLModel simplifies the project by allowing a single model definition to serve as both the database schema and FastAPI response model. Streamlit enables the rapid development of a clean, Python-based interactive interface, eliminating the need to build HTML/JS pages and maintaining focus on data analysis and Industrial Engineering decision support.