ChatGPT

RightNow NYC – Project README & Team Plan

Project Overview: RightNow NYC is a web app that visualizes live New York City 311 complaints on an interactive map and highlights permitted protests/events. Users can filter and cluster 311 calls in real time, and see official Street Activity Permit Office (SAPO) events overlaid on the city map. The app answers questions like “Where are noise complaints spiking?” or “Which protests are permitted downtown this week?” by querying NYC Open Data APIs and rendering results in the UI.

Key Features:
- **Real-Time 311 Map:** Displays up-to-date 311 service requests on a NYC map. Supports filtering by complaint type, borough, date range, and status. Uses spatial clustering so closely-packed complaints aggregate into summary markers. Provides analytics (e.g. counts by type) for the filtered data.
- **Permitted Events Layer:** Shows upcoming permitted protests/events from the SAPO dataset. This layer updates daily and can be toggled on/off. Users can click an event to see its details (event name, date/time, permit info).
- **Interactive Filters and Analytics:** Side panels let users select filters (e.g. “Noise”, “Sanitation”, date range, borough). The app recalculates and re-renders points and clusters on the fly. An analytics dashboard (charts or summaries) updates to show totals or trends.
- **Backend Data Integration:** A Flask-based API fetches data from NYC Open Data endpoints, applies queries and caching, and serves JSON to the frontend.

Architecture & Workflow

RightNow NYC is structured as a full-stack web app:
- **Data Backend (Python/Flask):** Queries the Socrata APIs for 311 and SAPO data, runs clustering, and caches results. Exposes REST endpoints that the frontend calls.
- **Frontend (React + Leaflet + Tailwind CSS):** Renders the interactive map and UI. Consumes backend endpoints to plot markers and show data. Tailwind CSS ensures a clean, responsive design.
- **Deployment/DevOps:** The app is containerized (Docker) and continuously deployed. Clustering and data pipelines run as scheduled tasks to preprocess large datasets.

Team Responsibilities

Ammaar – Lead Architect / DevOps / Backend Clustering

Summary: Ammaar designs the overall system, builds data pipelines, and implements geospatial clustering. He ensures the app runs reliably in production (CI/CD, Docker/Kubernetes).

- **Tasks:**
- **Data Ingestion & Preprocessing:** Write Python scripts to periodically fetch raw 311 and SAPO data (via Socrata API), extract coordinates, and clean records.

- **Clustering:** Implement spatial clustering on 311 complaint coordinates to reduce map clutter. For example, using scikit-learn's KMeans:

```
from sklearn.cluster import KMeans

coords = [(40.7128, -74.0060), (40.7138, -74.0050), ...]  # (lat, lon)
points
kmeans = KMeans(n_clusters=10, init='k-means++', random_state=42)
labels = kmeans.fit_predict(coords)
# labels[i] gives cluster index for coords[i]
```

This assigns each complaint to a cluster. Centroids can be returned to the frontend as cluster markers.

- **DevOps & Deployment:** Create Dockerfiles and CI pipelines. Configure the Flask app on a server or Kubernetes. Ensure environment variables (API tokens) are secure. Implement health checks and logging.

- **APIs & Usage:** Ammaar's scripts query:

- **NYC 311 API:** via `https://data.cityofnewyork.us/resource/erm2-nwe9.json` [1]. E.g. using requests or `sodapy` with filters for date or location.
- **SAPO API:** via `https://data.cityofnewyork.us/resource/tvpp-9vvx.json` (NYC Permitted Event Information) [2]. He may filter by event date or use a bounding box.

Usage note: Socrata's API supports SoQL queries. For example, to get recent 311 calls:

```
https://data.cityofnewyork.us/resource/erm2-nwe9.json?$where=created_date>=
'2025-10-01T00:00:00'
```

Authentication via an application token (passed in headers `X-App-Token`) allows higher request limits [3] [4].

David – Backend Developer (Flask API / Data Flow)

Summary: David builds the Flask API server. He writes routes that the React frontend calls to fetch filtered data. He handles query logic, caching, and error management.

- **Tasks:**

- **API Endpoints:** Define Flask routes for various data needs. For example, `/api/complaints` could return 311 complaints filtered by query parameters (date range, type, bbox). Example route:

```
from flask import Flask, request, jsonify
import requests
```

```
app = Flask(__name__)

@app.route('/api/complaints')
def get_complaints():
    # Read query params
    complaint_type = request.args.get('type')
    bbox = request.args.get('bbox')  # e.g. "40.7,-74.02,40.8,-73.95"

    # Build SoQL query
    socrata_url = "https://data.cityofnewyork.us/resource/erm2-nwe9.json"
    socrata_params = {"$limit": 1000}
    if complaint_type:
        socrata_params["$where"] = f"complaint_type='{complaint_type}'"
    if bbox:
        lat1,lon1,lat2,lon2 = bbox.split(',')
        socrata_params["$where"] = (socrata_params.get("$where","") +
            f" AND within_box(location,{lat1},{lon1},{lat2},{lon2})")

    # Fetch from NYC API
    response = requests.get(socrata_url, params=socrata_params,
headers={"X-App-Token": SODA_TOKEN})
    data = response.json()
    return jsonify(data)
```

This example fetches and returns filtered 311 complaints.

- **Data Integration:** Call Socrata endpoints within routes. Process and format data as needed (e.g. compute cluster groups, rename fields).

- **Filtering & Pagination:** Parse incoming query parameters to build SoQL filters (`$where`, `$select`). Use `$limit` and `$offset` to page through large results if necessary [5]. E.g. loop with offset for more than 1000 rows.
- **Caching:** Implement caching (e.g. in-memory or Redis) so repeated queries don't always hit the NYC API. Cache results of expensive queries (like full-bounds requests) for e.g. 5–10 minutes.

- **Error Handling:** Validate inputs and catch API errors. Return HTTP 400 for bad requests or 500 on server errors. Example:

```
@app.errorhandler(400)
def bad_request(e):
    return jsonify(error="Bad request"), 400
```

- **APIs & Usage:** David's code queries:

- **NYC 311 Resource:** `erm2-nwe9.json` as above [1]. He may use a combination of SoQL filters. For example, to get complaints in Manhattan:

```
$where=borough='MANHATTAN'&$limit=500
```

- **SAPO Resource:** e.g. `tvpp-9vvx.json`. He might query upcoming events:

```
$where=event_start >= '2025-10-01T00:00:00'
```

He converts the SAPO data into a form (e.g. list of `{lat, lon, name, date}`) that the frontend can plot.

- **Code Snippet (Flask route):**

```
from flask import Flask, request, jsonify
import requests

app = Flask(__name__)

@app.route('/api/events')
def get_events():
    # Fetch SAPO events from NYC API
    socrata_url = "https://data.cityofnewyork.us/resource/tvpp-9vvx.json"
    params = {"$where": "event_start >= '2025-10-01T00:00:00'"}
    resp = requests.get(socrata_url, params=params)
    events = resp.json()
    # Transform data
    simplified = [{"name": e["event_name"], "lat": float(e["latitude"]),
                   "lon": float(e["longitude"]), "start": e["event_start"]}
for e in events]
    return jsonify(simplified)
```

Nikhil – Frontend Developer (Map Integration)

Summary: Nikhil creates the interactive NYC map in React. He loads the map view, fetches data from David's API, and plots markers and clusters.

- **Tasks:**
- **Map Component:** Use React Leaflet to render a map of NYC. Center on Manhattan by default (e.g. latitude 40.7128, longitude -74.0060) [6].
- **Plot Data Points:** When the backend returns complaint data, add `<Marker>` elements at each point's coordinates. Use `<Popup>` to show complaint details on click.
- **Clustering (frontend):** Optionally use a Leaflet marker cluster plugin (or pre-clustered points from backend). Manage re-render when new data arrives (due to filter changes).

- **Filters Integration:** Listen for filter controls (type, date, etc.) and call the corresponding API endpoint. For example:

```
function MapView() {
  const [complaints, setComplaints] = useState([]);
  useEffect(() => {
    fetch(`/api/complaints?type=Noise&borough=QUEENS`)
      .then(res => res.json())
      .then(data => setComplaints(data));
  }, [filterType, filterBorough]);

  return (
    <MapContainer center={[40.7128, -74.0060]} zoom={12}>
      <TileLayer
        attribution='© OpenStreetMap'
        url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
      />
      {complaints.map((c, idx) => (
        <Marker key={idx} position={[c.latitude, c.longitude]}>
          <Popup>
            <strong>{c.complaint_type}</strong><br/>
            {c.descriptor}
          </Popup>
        </Marker>
      ))}
    </MapContainer>
  );
}
```

Example React-Leaflet map with markers [6] .

- **APIs & Usage:** Nikhil's code calls:

- **Backend Endpoints:** e.g. `GET /api/complaints?...` and `GET /api/events`.
- **Leaflet Map Tiles:** Using OpenStreetMap via `<TileLayer url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png" />`.
- **Clustering Libraries:** Could use react-leaflet-markercluster for grouping markers on the map.

Ethan – Frontend Developer (UI/UX & Demo)

Summary: Ethan designs the user interface with React and Tailwind CSS. He builds layouts (filters sidebar, info panels) and styles components. He also prepares the demo and final polish.

- **Tasks:**
- **UI Components:** Create reusable React components for filter controls (dropdowns, date pickers), info cards, and navigation. For example, a complaint card:

```
function ComplaintCard({ complaint }) {
  return (
    <div className="p-4 bg-white shadow rounded">
      <h3 className="text-lg font-bold">{complaint.complaint_type}</h3>
      <p className="text-sm text-gray-600">{complaint.descriptor}</p>
      <p className="text-xs text-gray-500">{complaint.created_date}</p>
    </div>
  );
}
```

- **Tailwind Styling:** Use utility classes for colors, spacing, and responsiveness. For example, a filter button:

```
<button className="bg-blue-500 hover:bg-blue-700 text-white font-bold py-2
px-4 rounded">
  Apply Filters
</button>
```

This button uses Tailwind classes for a blue background and hover effect [7] .

- **Responsive Layout:** Ensure the app works on mobile/tablet by using Tailwind's responsive prefixes (e.g. `md:flex-row`).
- **Demo Preparation:** Assemble a scripted demo walkthrough. Ensure each feature is showcased (e.g. toggling events, adjusting filters, displaying analytics). Polish copy and visuals for clarity.
- **APIs & Usage:** Ethan's UI components consume the same backend APIs. He also ensures any required API endpoints (e.g. for analytics summaries) are connected. No new external APIs beyond React/Leaflet/Tailwind.

API & Data Details

- **NYC 311 Open Data API:** We use the "311 Service Requests" dataset (ID `erm2-nwe9`) on data.cityofnewyork.us [1] . It can be queried via:

```
https://data.cityofnewyork.us/resource/erm2-nwe9.json
```

Key points:

- **SoQL Filtering:** Use `$where` to filter. E.g. `? $where=complaint_type='Noise' AND created_date >= '2025-01-01T00:00:00'` retrieves noise complaints this year. Socrata's SoQL is SQL-like [8] .
- **Bounding Box:** To fetch only points in the current map view, use `within_box(location, lat1, lon1, lat2, lon2)` in `$where`. For example:

```
$where=within_box(location, 40.9176, -74.2591, 40.4774, -73.7004)
```

(Coordinates for NYC's northeast and southwest corners.) Socrata provides functions like `within_box()` for spatial queries.

- **Pagination:** Socrata defaults to 1000 rows. Use `$limit` and `$offset` to page (e.g. `$limit=1000&$offset=1000`). The Flask backend handles this transparently by aggregating pages if needed.

- **Error Handling:** If the API call fails or returns empty, the Flask route returns a JSON error with an appropriate HTTP status. We validate query inputs (dates in ISO format, numeric bounding values) and return 400 on invalid inputs.

- **SAPO Permitted Event API:** We use NYC's "NYC Permitted Event Information" dataset (ID `tvpp-9vvx`) [2], which lists street-event permits for the coming month. Access via:

```
https://data.cityofnewyork.us/resource/tvpp-9vvx.json
```

Example usage: filter by date range or type. E.g. `$where=event_start >= '2025-10-15' AND event_end <= '2025-10-30'`. We extract each event's latitude/longitude for mapping. Event names and dates populate popups.

- **SoQL Query Examples:**

- **Date Range Filter:** `https://data.cityofnewyork.us/resource/erm2-nwe9.json?$where=created_date >= '2025-10-01'`.
- **Complaint Type:** `$where=complaint_type='HEAT/HOT WATER'`.
- **Bounding Box:** as above.
- **Aggregation:** For analytics, use `$select` and `GROUP BY`. E.g. to count complaints by borough:

```
https://data.cityofnewyork.us/resource/erm2-nwe9.json?$select=borough,
count(unique_key)&$group=borough
```

- **Sort/Limit:** `$order=created_date DESC&$limit=100`.

- **Caching Strategy:** The backend caches frequent queries (e.g. default full-city view) for a short TTL (e.g. 10 minutes). We use Python's `functools.lru_cache` or Redis to store recent results. This avoids exhausting Socrata's rate limits. The app token header and caching together ensure performance.

Demo Script (Summary)

1. **Launch App:** User navigates to the RightNow NYC web app. By default, the NYC map loads with all 311 complaints from the past 24h (clustered by neighborhood).

2. **Inspect Complaints:** Hover/click a cluster to zoom in. Notice individual pins appear. Click a pin to see complaint details (type, time, description).
3. **Apply Filters:** In the sidebar, select “Complaint Type: Noise” and borough “MANHATTAN”. The map refreshes: clusters update to show only Manhattan noise complaints [5]. The analytics panel now shows total count and trend chart (if implemented).
4. **Show Events Layer:** Toggle “Upcoming Events” on. Red markers (or icons) appear for permitted rallies/parades from the next week (pulled from SAPO) [2]. Click an event to read its name and time.
5. **Spatial Intersection:** Pan/zoom to Midtown. Observe any complaints near an event marker. This could hint at related 311 calls.
6. **Analytics:** Open the dashboard. See bar graphs: e.g., top 5 complaint types in selected area. Demonstrate a SoQL aggregation (counts by type) and how backend returned it.
7. **Responsive Check:** Resize the browser to tablet width. Confirm the layout adjusts (menu collapses, map remains interactive).
8. **Conclusion:** All features (map, filters, events, clustering, analytics) are working as intended.

Sprint Checklist (by Oct 30)

- **Week 1 (Oct 7–13):**
- [ ] **Map Setup:** Nikhil integrates React Leaflet, displays base NYC map and sample markers [6].
- [ ] **Backend Skeleton:** David sets up Flask project with a test “hello world” route.

- [ ] **Data Fetching (DEV):** Ammaar writes a script to fetch 1000 311 calls via Socrata API (using `requests` or `sodapy`) and prints summary.

- **Week 2 (Oct 14–20):**

- [ ] **Clustering Logic:** Ammaar implements KMeans clustering on 311 coordinates and exposes cluster centers via an API.
- [ ] **Complaint Endpoint:** David creates `/api/complaints` endpoint (with filters for date/type) and tests it.
- [ ] **Map Markers:** Nikhil fetches `/api/complaints` in React and plots points for each complaint [6].

- [ ] **Event Endpoint:** David and Ammaar set up `/api/events` pulling SAPO data. Ethan designs the toggle button to enable this layer.

- **Week 3 (Oct 21–27):**

- [ ] **Clustering UI:** Nikhil adds marker clustering plugin for readability at low zooms.
- [ ] **Filters UI:** Ethan implements filter controls (dropdowns, datepickers) in React and wires them to API queries.
- [ ] **Analytics Panel:** David implements an endpoint (e.g. `/api/stats`) that runs SoQL aggregation (counts by type). Ethan renders this in a chart (e.g. using Chart.js or React-Vis).

- [ ] **Styling & Responsiveness:** Ethan applies Tailwind classes site-wide and tests on different screen sizes.

- **Week 4 (Oct 28–30):**

- [ ] **Testing & QA:** Team tests all features together. Fix bugs (e.g. API errors, misaligned UI).
- [ ] **Demo Prep:** Ethan finalizes slide deck/script. Nikhil and David prepare to demo map interactions.
- [ ] **Deployment:** Ammaar finalizes Docker setup, deploys staging version, and ensures logs/monitoring are active.

All tasks marked complete and reviewed by Oct 30.

References

- NYC Open Data Portal – *311 Service Requests from 2010 to Present* (`erm2-nwe9`) [1] and *NYC Permitted Event Information* (`tvpp-9vvx`) [2].
- Socrata Developer Documentation – SODA Query Language (SoQL) and API usage [8].
- React Leaflet documentation (map and marker example) [6].
- Tailwind CSS examples for utility classes [7].
- StackOverflow – example SoQL queries on NYC 311 dataset [4] (filter by date).

---

[1] Metadata and queries for open data
https://geographyplanning.buffalostate.edu/wMix/python/Metadata%20and%20queries%20for%20open%20data.html

[2] NYC Permitted Event Information - Catalog
https://catalog.data.gov/dataset/nyc-permitted-event-information

[3] [8] Queries using SODA3 | Socrata
https://dev.socrata.com/docs/queries/

[4] [5] json - Using Socrata SODA API to query most recent rows by datetime - Stack Overflow
https://stackoverflow.com/questions/25001635/using-socrata-soda-api-to-query-most-recent-rows-by-datetime

[6] Popup with Marker | React Leaflet
https://react-leaflet.js.org/docs/v3/example-popup-marker/

[7] A Simple Guide to Background Colors in Tailwind CSS | Tailkits
https://tailkits.com/blog/tailwind-bg-color-guide/