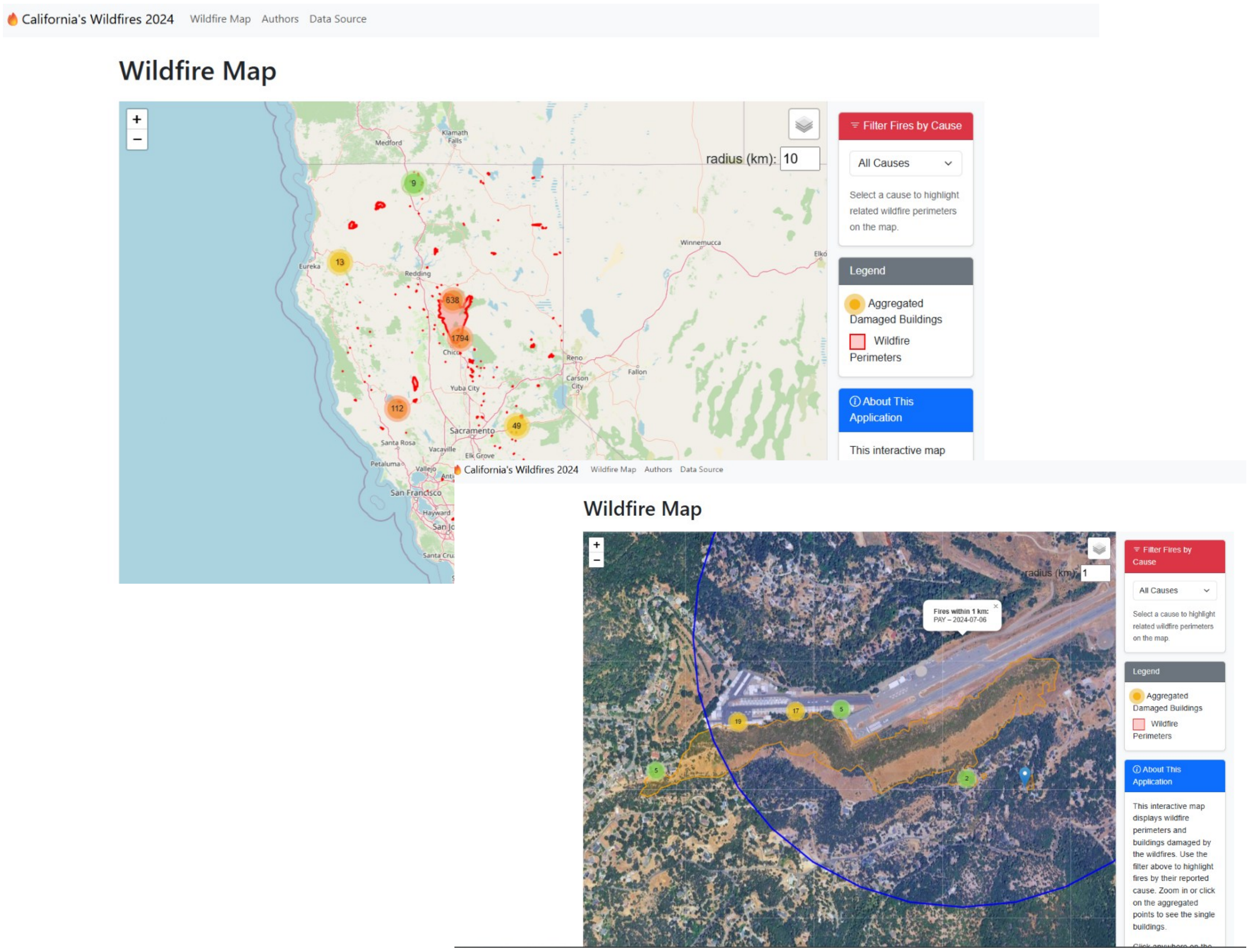
Lisa Glahsl | Anne Kahles

# Final Report: A Web Application to Explore and Analyze 2024 California Wildfire Data

Our goal is to create an interactive map for wildfires in California, USA. This webmap shows the locations of the wildfires and its fire perimeters for the year 2024. Additionally the user is able to engage with the Website, for example create bufferzones to show which residential areas were affected, get further information about individual fires and about damaged buildings.

Implementation: Web-GIS, Django Python, JavaScript, HTML, CSS (Bootstrap)



Select a cause to highlight related wildfire perimeters on the map.

This interactive map

Select a cause to highlight related wildfire perimeters on the map

This interactive map displays wildfire perimeters and buildings damaged by the wildfires. Use the filter above to highlight fires by their reported cause. Zoom in or click on the aggregated points to see the single buildings.

## 1 State of the Art

Wildfire (extent and occurrence) Dashboards are often based on satellite images and provide real-time data for active fire sources. There are also studies analyzing the wildfire occurrence over the past decades and providing fire frequencies as heatmaps (clustering). Other tools such as WIFIRE Firemap provide dynamic modelling of current fire development, interactive risk-simulation-Web-GIS or machine-learning supported products.

## 2 Problem Statement

There is an increase of temperature due to climate change, which also lead to an increase of wildfire incidents worldwide. California provides several interactive Dashboard for private users to look at historical and current fire events (see Dashboard CAL FIRE). The data can only be explored visually and the user interface is a bit confusing. In addition, the Dashboard Product from ESRI was used, which is proprietary.

For non-expert users such as local residents the solutions described in the State of the Art part are sometimes a bit complex and hard to understand.

## 3 Objective

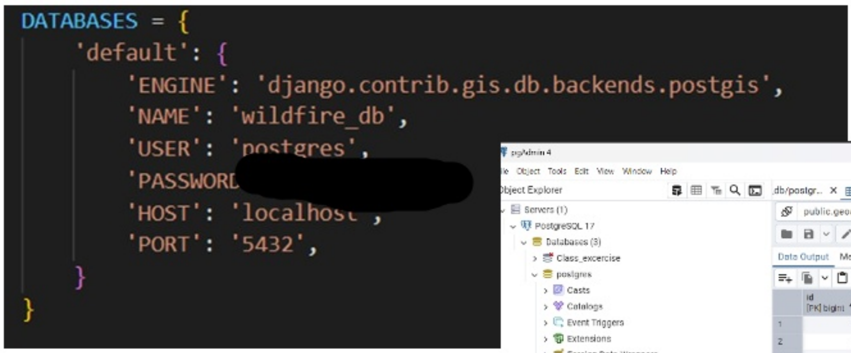
This leads us to the following research question: How to provide fire data from 2024 for the state of California in an interactive user-friendly WebGIS application, enabling users to explore fire locations, causes, affected areas, and containment dates based on open-source products? The application aims to allow users to filter wildfire data dynamically, visualize spatial relationships, and perform proximity analyses directly on the map. A key feature is the ability for users to define a custom search radius to identify all wildfires intersecting a given buffer zone around a clicked location. By implementing these tools in a web-based environment using Leaflet, Turf.js, and GeoJSON, the project seeks to demonstrate how spatial data can be interactively queried and analyzed without the need for desktop GIS software.

## 4 Methods

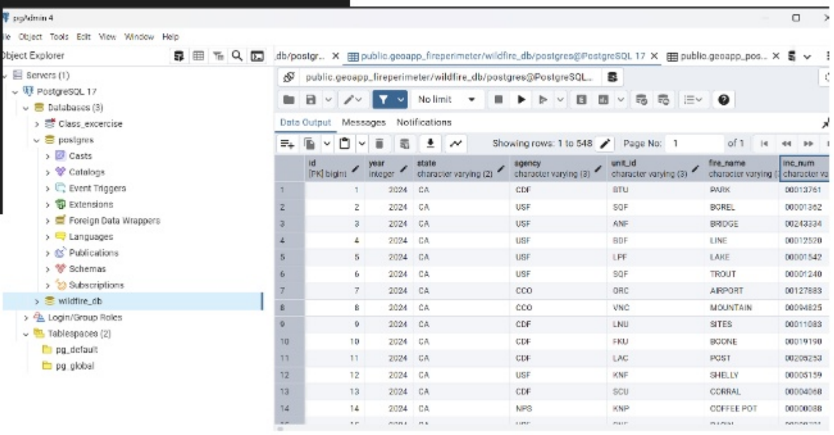
The project is implemented as a browser-based WebGIS application using JavaScript, Leaflet, and Turf.js for the frontend, combined with a Python/Django backend for data management. During our project we went through the following main steps:

### Backend (Python / Django)

1. **Data storage** – Storing wildfire data in a PostGIS-enabled PostgreSQL database (wildfire_db) for efficient spatial queries.

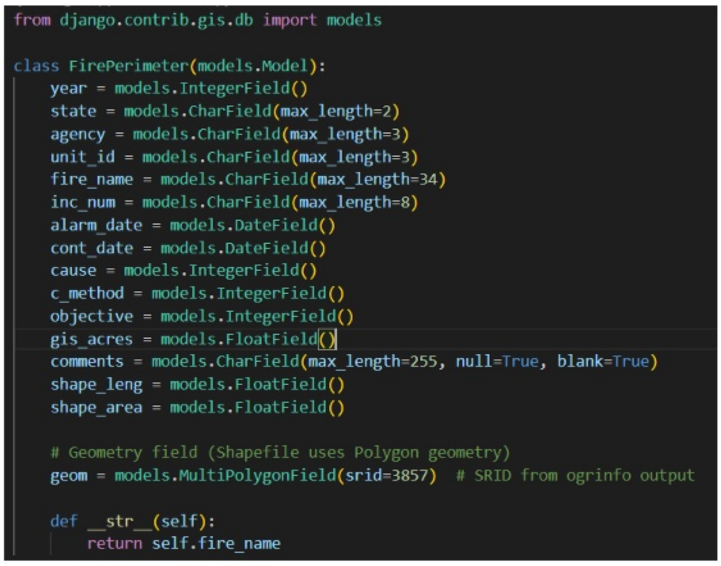


```python
DATABASES = {
    'default': {
        'ENGINE': 'django.contrib.gis.db.backends.postgis',
        'NAME': 'wildfire_db',
        'USER': 'postgres',
        'PASSWORD[redacted]
        'HOST': 'localhost',
        'PORT': '5432',
    }
}
```



| | id [PK] bigint | year integer | state character varying (2) | agency character varying (3) | unit_id character varying (3) | fire_name character varying | inc_num character va |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 2024 | CA | CDF | BTU | PARK | 00013761 |
| 2 | 2 | 2024 | CA | USF | SQF | BOREL | 00001362 |
| 3 | 3 | 2024 | CA | USF | ANF | BRIDGE | 00243334 |
| 4 | 4 | 2024 | CA | USF | BDF | LINE | 00012520 |
| 5 | 5 | 2024 | CA | USF | LPF | LAKE | 00001542 |
| 6 | 6 | 2024 | CA | USF | SQF | TROUT | 00001240 |
| 7 | 7 | 2024 | CA | CCO | ORC | AIRPORT | 00127883 |
| 8 | 8 | 2024 | CA | CCO | VNC | MOUNTAIN | 00094825 |
| 9 | 9 | 2024 | CA | CDF | LNU | SITES | 00011083 |
| 10 | 10 | 2024 | CA | CDF | FKU | BOONE | 00019190 |
| 11 | 11 | 2024 | CA | CDF | LAC | POST | 00208253 |
| 12 | 12 | 2024 | CA | USF | KNF | SHELLY | 00005159 |
| 13 | 13 | 2024 | CA | CDF | SCU | CORRAL | 00004068 |
| 14 | 14 | 2024 | CA | NPS | KNP | COFFEE POT | 00000088 |

2. **Data preparation & preprocessing** – Creating a geoapp, and within this models for our two datasets: wildfire perimeters (polygons) and postfire damaged buildings (points)



```python
from django.contrib.gis.db import models

class FirePerimeter(models.Model):
    year = models.IntegerField()
    state = models.CharField(max_length=2)
    agency = models.CharField(max_length=3)
    unit_id = models.CharField(max_length=3)
    fire_name = models.CharField(max_length=34)
    inc_num = models.CharField(max_length=8)
    alarm_date = models.DateField()
    cont_date = models.DateField()
    cause = models.IntegerField()
    c_method = models.IntegerField()
    objective = models.IntegerField()
    gis_acres = models.FloatField()
    comments = models.CharField(max_length=255, null=True, blank=True)
    shape_leng = models.FloatField()
    shape_area = models.FloatField()

    # Geometry field (Shapefile uses Polygon geometry)
    geom = models.MultiPolygonField(srid=3857)  # SRID from ogrinfo output

    def __str__(self):
        return self.fire_name
```

Pre-processing on the server side also ensures attributes are cleaned and standardized (e.g., mapping cause codes to readable descriptions) before sending them to the frontend.

3. **API endpoints** – Creating Django REST Framework (DRF) endpoints, such as /api/wildfires/, returning data in GeoJSON format. Filtering parameters (e.g., cause=, start_date=) can be passed via query strings

4. **Integration with Leaflet** – The frontend fetches GeoJSON from these endpoints using fetch() calls and dynamically updates the map layer without reloading the page.

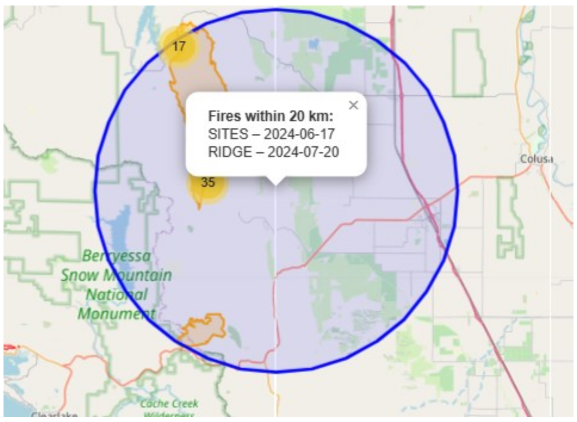
**Frontend (JavaScript / Leaflet / Turf.js)**

The Frontend structure is made of an authors.html, a base.html where the base structure is configured, a map.html, where we linked all documents together, as well as styling and configuring the legend, sidebar and infocard. Our JavaScript file called map.js contains most of the code connecting backend and frontend.

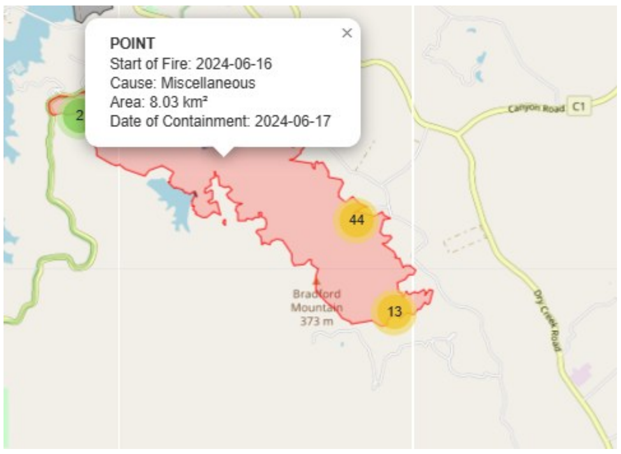
1. **Map setup** – Leaflet was used to create the open-source base map layers, load wildfire polygons, damage points and style them dynamically.

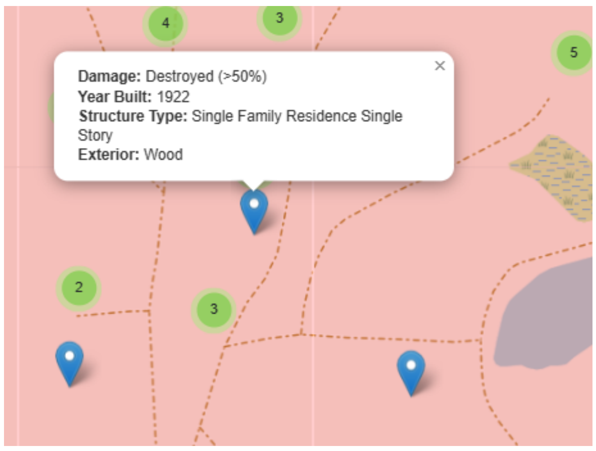
2. **Filtering functionality** – A dropdown menu allows filtering by wildfire cause, updating the map layer in real time.

3. **Proximity analysis tool** – A custom Leaflet control was implemented to let the user input a radius (in km). When clicking on the map, Turf.js will generate a buffer around the click point, and all intersecting wildfires will be identified and highlighted.


Fires within 20 km:
SITES – 2024-06-17
RIDGE – 2024-07-20

4. **User interaction** – For each intersecting fire, a popup displays the key attributes name of the fire and alarm date. Non-intersecting fires will revert to default styling. When clicking on wildfires or mapped buildings in general, according popups provide information.


POINT
Start of Fire: 2024-06-16
Cause: Miscellaneous
Area: 8.03 km²
Date of Containment: 2024-06-17


**Damage:** Destroyed (>50%)
**Year Built:** 1922
**Structure Type:** Single Family Residence Single Story
**Exterior:** Wood

5. **Testing and validation** – The tool was be tested using various input radii and click locations to ensure the analysis logic returns correct results. We also showed the map to non-expert users unfamiliar with GIS or cartography and implemented their feedback, such as an information text for better guidance.

By combining these components, the project provides an accessible and interactive way to query spatial wildfire data, showcasing the potential of WebGIS for environmental hazard exploration.

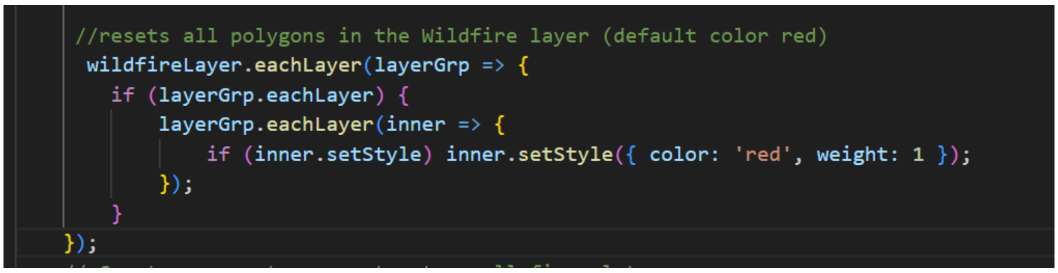
## 5 Results (Problem Solution)

Problems we faced and how we solved them:

- **Click Event Buffer**: The first spatial buffer we implemented took the center of each wildfire polygon to calculate the proximity. That resulted in non-realistic outcomes as the polygons are not perfect circles. So when clicking right next to a fire, it sometimes did not recognize the fire as within the buffer, as while logically the fire was located in the set radius.
  **Solution**: First, a Turf.js point is created with turf.point([lng, lat]). We then built a buffer around the clickpoint. All Wildfire polygons are then checked for overlaps with this circle (turf.booleanIntersects), based on their geometry. The

buffer is now applied directly to the feature object with the radius now referring to the edge of the burn area and not to an arbitrary centre point.

- **Resetting wildfire styling after analysis:** After running a proximity analysis, highlighted features stayed orange even when the next query was made, leading to visual confusion.
  **Solution**: Added a pre-analysis loop that resets all wildfire features back to their default red style before applying new highlights:



```
 //resets all polygons in the Wildfire layer (default color red)
  wildfireLayer.eachLayer(layerGrp => {
    if (layerGrp.eachLayer) {
        layerGrp.eachLayer(inner => {
            if (inner.setStyle) inner.setStyle({ color: 'red', weight: 1 });
        });
    }
});
```

- **Efficient server-side filtering:** Initially, all data from the database was loaded into the frontend, which affected the performance of our Web-App. Specifically the damaged buildings dataset slowed down the performance because it contained over 8,000 points.
  **Solution:** Embedding the Leaflet Plug-in L.markerClusterGroup() providing Clustering of spatially close points.

## Contribution

| **Task** | **Main contribution by group member** |
|---|---|
| Data Collection | Lisa, Anne |
| Integrating data into database | Anne, Lisa |
| Connecting backend and frontend with API | Anne, Lisa |
| Setting up Web-Page-Structure | Lisa |

| | |
|---|---|
| Adding functionality in JavaScript | Anne, Lisa |
| Final Report | Anne |

## Links

**Our GitHub Repository**:

https://github.com/lisogla/Final_Project_Application_Development_Wildfires/tree/master

On GitHub we also included some screenshots of our final Web-Application and its functionality.

**The resulting Web-Application**:

http://127.0.0.1:8000/api/map/

## Generative AI Statement:

We used generative AI, such as ChatGPT or Google Gemini to perfect our code, as well as when we got stuck in some places. We did not simply accept AI prompts without checking them, but tried to understand them line by line and incorporate them in the right places. Example questions we asked AI to help with our project:

ChatGPT:

- "What does this error message mean in the console: map/:1 Uncaught (in promise) SyntaxError: Unexpected token "<", '<!DOCTYPE '... is not valid JSON"
- "We have connected a GeoJSON via API, which displays individual buildings that were damaged during the fires as points. Since the data set is very large, it causes the page to crash in some cases. Is there any way we can load the data in a more compact format?"