AQI Metric Prediction System Using Machine Learning and Full-Stack Cloud Deployment

CONTENTS

CONCLUSION

- This project demonstrates a robust AQI monitoring and prediction platform that integrates real-time geospatial mapping, machine learning, and secure cloud deployment.
- The system is scalable and extensible for other smart city applications.

PROJECT OVERVIEW

Proposes a scalable, interactive Air Quality Index (AQI) prediction system using machine learning and cloud deployment.

Integrates modern frontend (React, Leaflet, Firebase) and backend (FastAPI, Docker, AWS) frameworks.

Uses Cloudflare Worker as a proxy for secure communication and to address CORS/XSS issues.

KEY FEATURES

- Real-time AQI prediction and visualization on interactive maps.
- User authentication via Google (Firebase).
- Users can upload custom AQI data (CSV) for analysis.
- Model selection:

Random Forest

Neural Networks,

Logistic Regression,

Gradient Boosting,

KNN.

- Dynamic graphs and histograms using Nivo, with site-specific predictions.

SYSTEM COMPONENTS

- SYSTEM ARCHITECTURE
- FLOW CHART
- FUNCTIONAL AND NON-FUNCTIONAL REQUIREMENTS
- USE CASE DIAGRM

SYSTEM ARCHITECTURE

- Frontend:

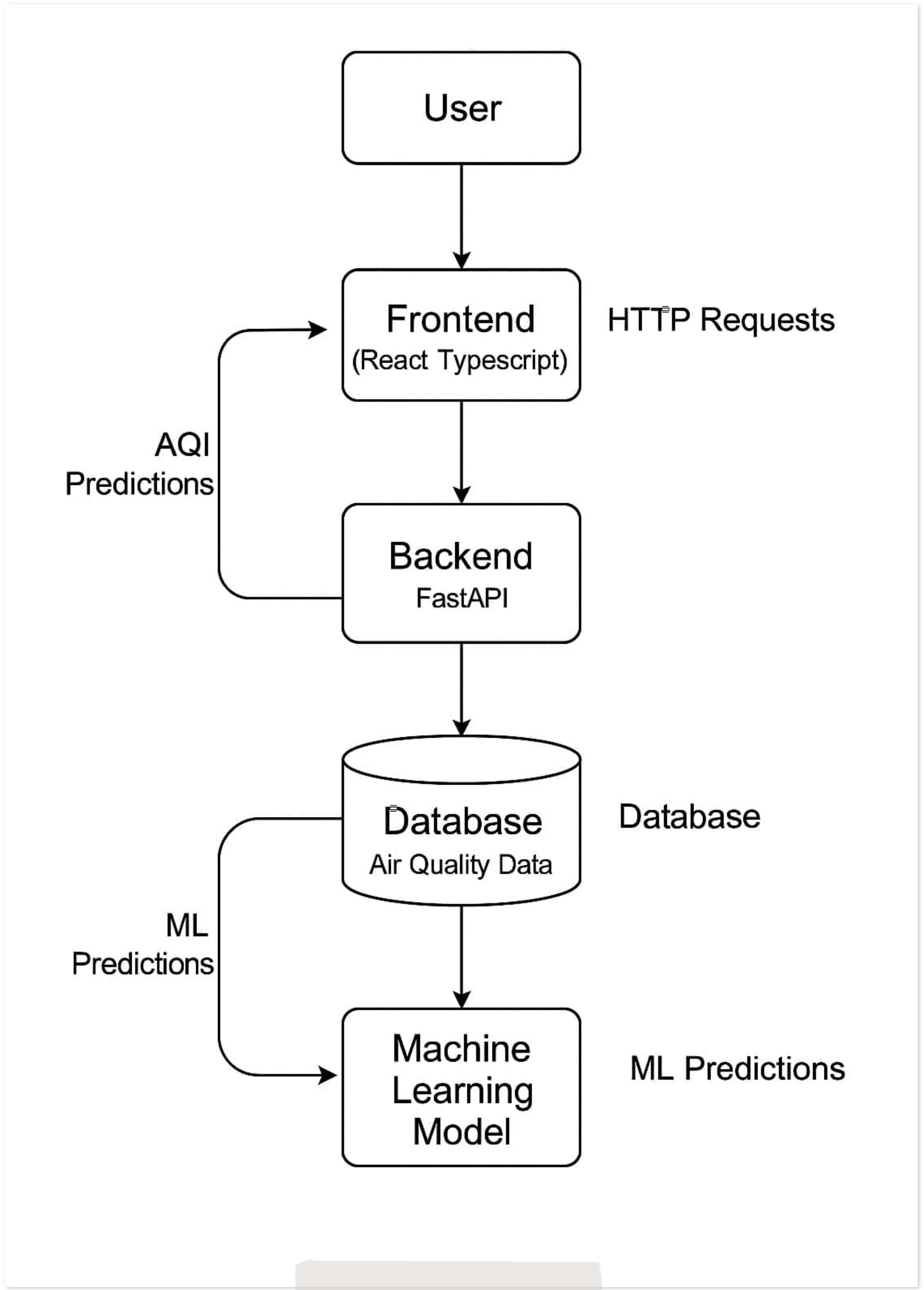
React (TypeScript, Material UI), Leaflet for OpenStreetMap, Firebase for authentication, Nivo for data visualization.

- Backend:

FastAPI (Python), SQLAlchemy ORM, Pydantic for validation, Dockerized deployment on AWS EC2.

- Proxy Layer:

Cloudflare Worker for secure API routing and protection against web vulnerabilities.

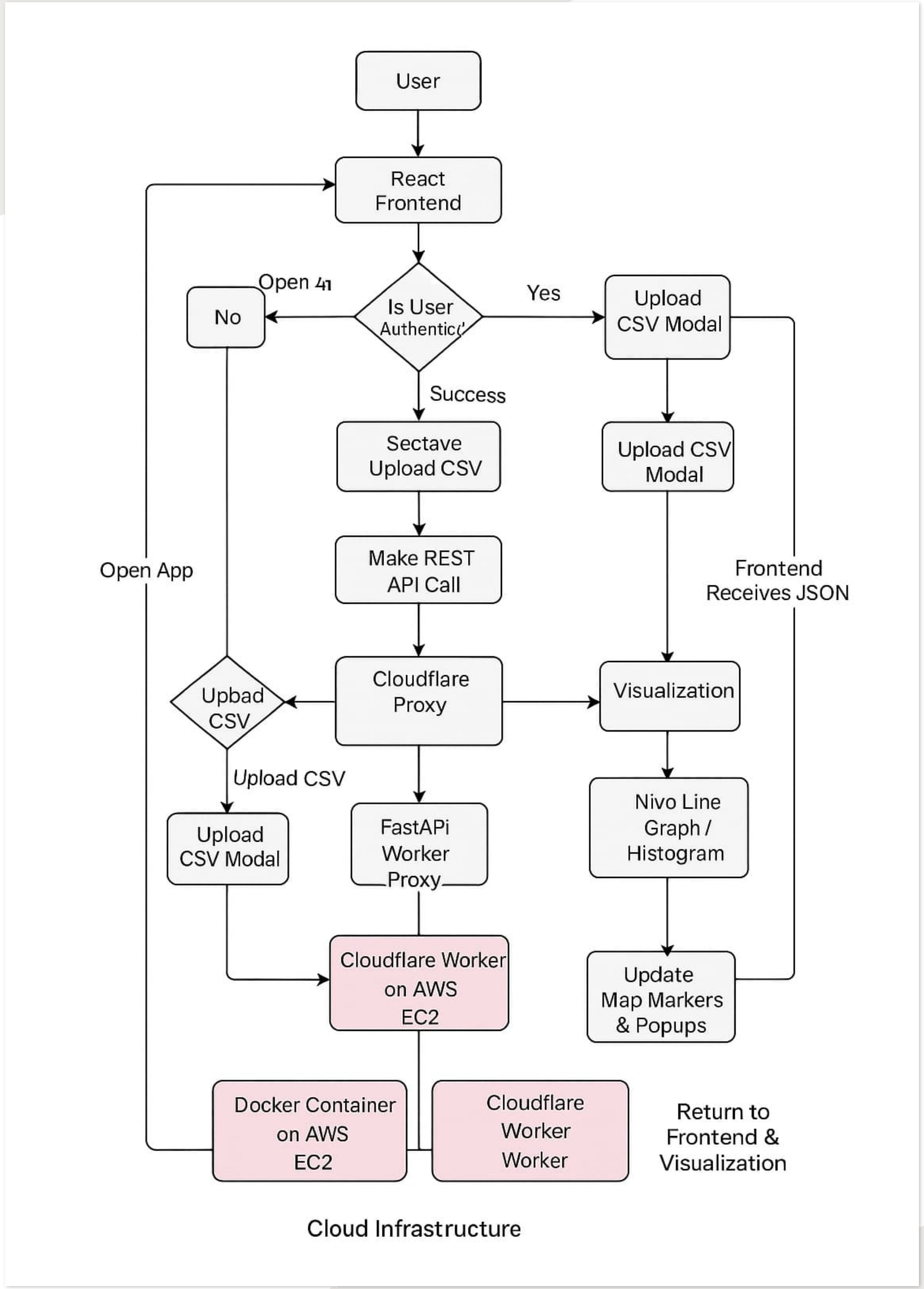


FLOW CHART OF THE UI

The flow chart shows a user interacting with a React frontend, which checks authentication before allowing CSV uploads.

Uploaded files are processed via Cloudflare proxy and backend services on AWS EC2, then returned as JSON.

The frontend uses this data for visualizations like line graphs and map updates.



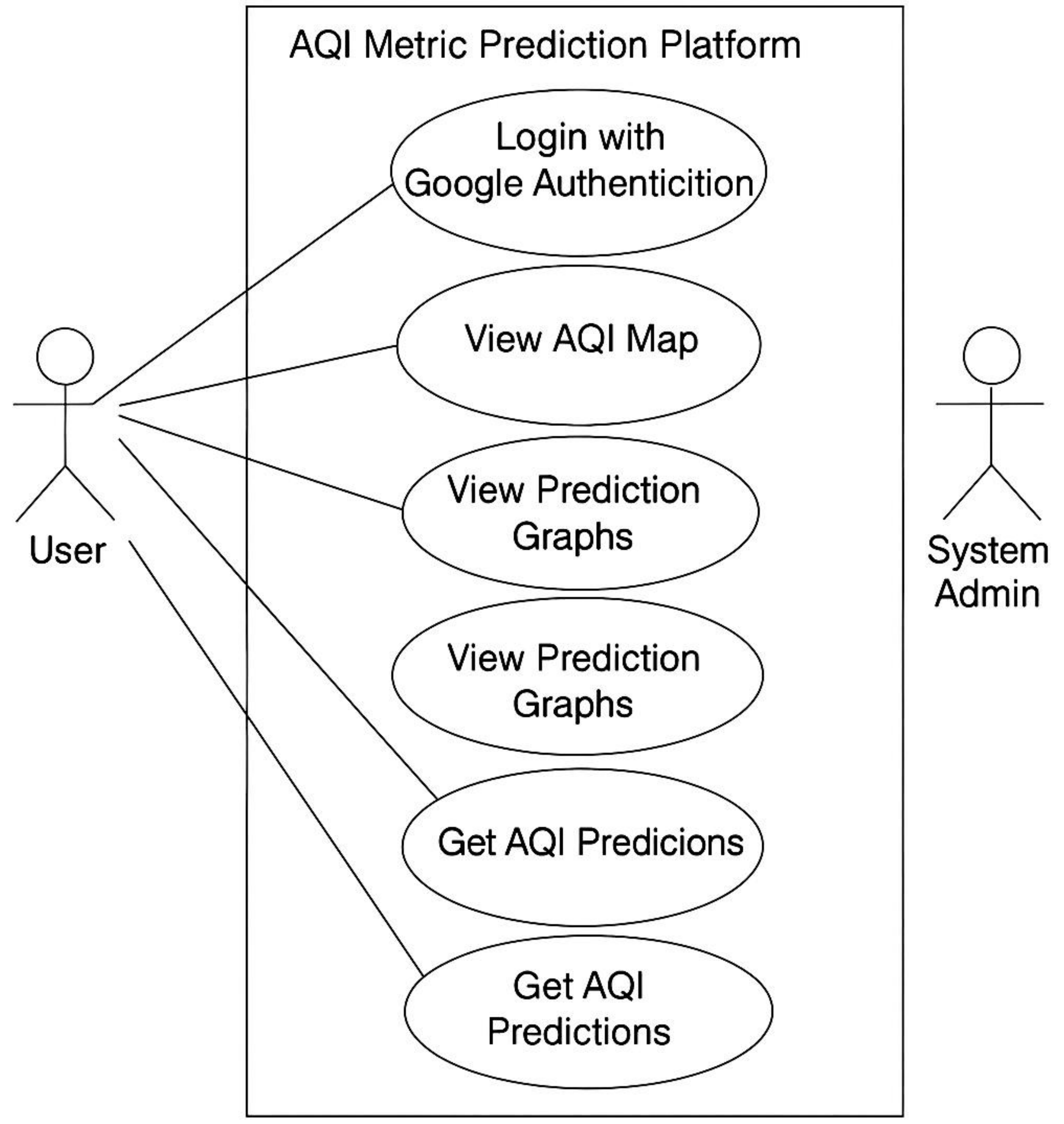
FUNCTIONAL AND NON-FUNCTIONAL REQUIREMENTS

FUNCTIONAL:

- User authentication and management.
- Interactive AQI data visualization on maps.
- Toggle between different ML models for prediction.
- CSV data upload and ingestion.
- Real-time prediction via RESTful APIs.
- Secure, proxied frontend-backend communication.

NON-FUNCTIONAL:

- High performance and low latency.
- Scalability for increased users/data (cloud-based).
- 24/7 reliability and minimal downtime.
- Maintainable, modular codebase.
- Portability across browsers/cloud platforms.
- Strong security for user data and APIs.
- Intuitive, user-friendly interface.



USE CASE DIAGRAM

- The diagram is a use case representation showing how users and system admins interact with the AQI Metric Prediction Platform.

Actors

- User: The primary actor who interacts with the platform to access AQI-related functionalities.
- System Admin: Shown as a secondary actor, typically responsible for backend management and system maintenance, though not directly linked to specific use cases in this diagram

User Functionalities

- Can Login with Google Authentication
- View AQI Map
- View ML calibrated Prediction Graphs:
- Get AQI Predictions

FRONT END COMPONENTS

- React - UI library for building web interfaces
- TypeScript - Strongly typed JavaScript
- Leaflet - Interactive maps for OpenStreetMap integration
- Firebase - Google authentication
- Axios - HTTP client for making API calls
- Nivo - Visualization library for graphs and charts React Context API

BACKEND COMPONENTS

FASTAPI - WEB FRAMEWORK FOR CREATING APIS

- SQLALCHEMY - ORM FOR DATABASE OPERATIONS

- PYDANTIC - DATA VALIDATION

- UVICORN - ASGI SERVER

- TENSORFLOW - FOR NEURAL NETWORK MODELS (.H5)

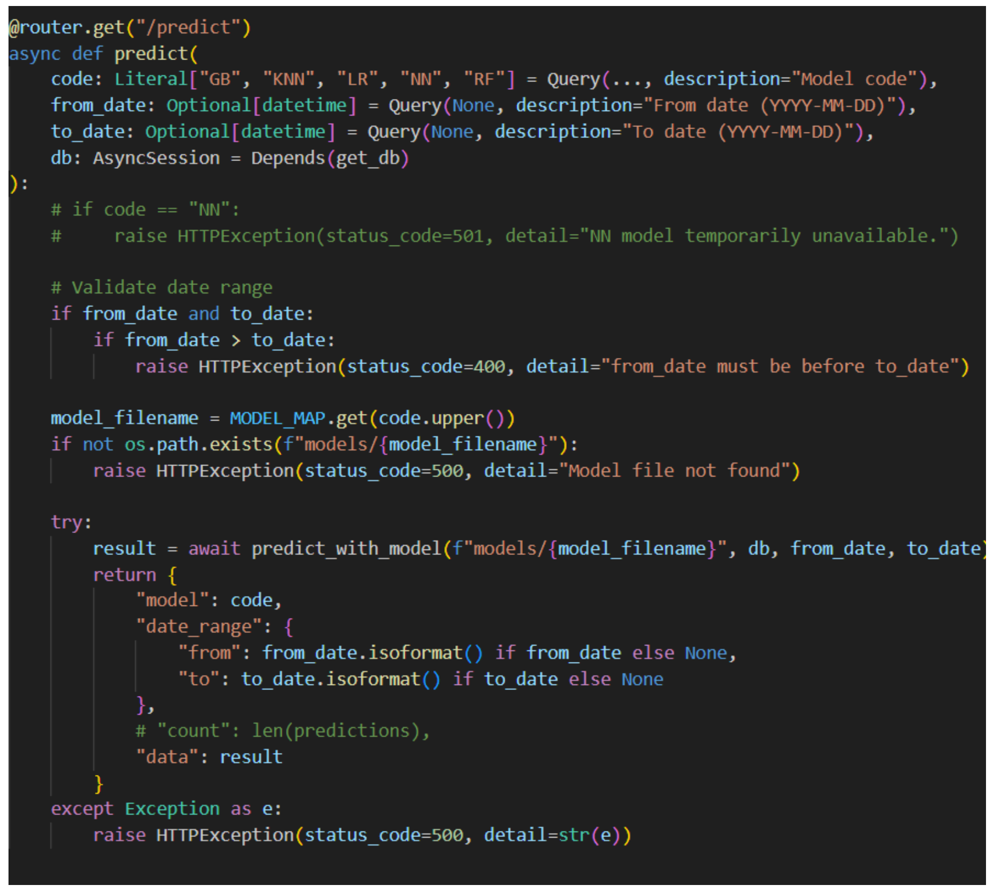
- JOBLIB - TO LOAD SCIKIT-LEARN MODELS

- SCIKIT-LEARN - ML MODEL FRAMEWORK

- PANDAS - DATA MANIPULATION

API END POINTS USED

- POST /air-quality-sites: Upload new AQI data.
- GET /air-quality-sites: Retrieve all AQI data.
- GET /air-quality-sites/region: Region-specific data.
- GET /metrics-filter: Filtered metrics.
- GET /all-regions: All regions' data.
- GET /predictors/predict: Model-based AQI prediction.

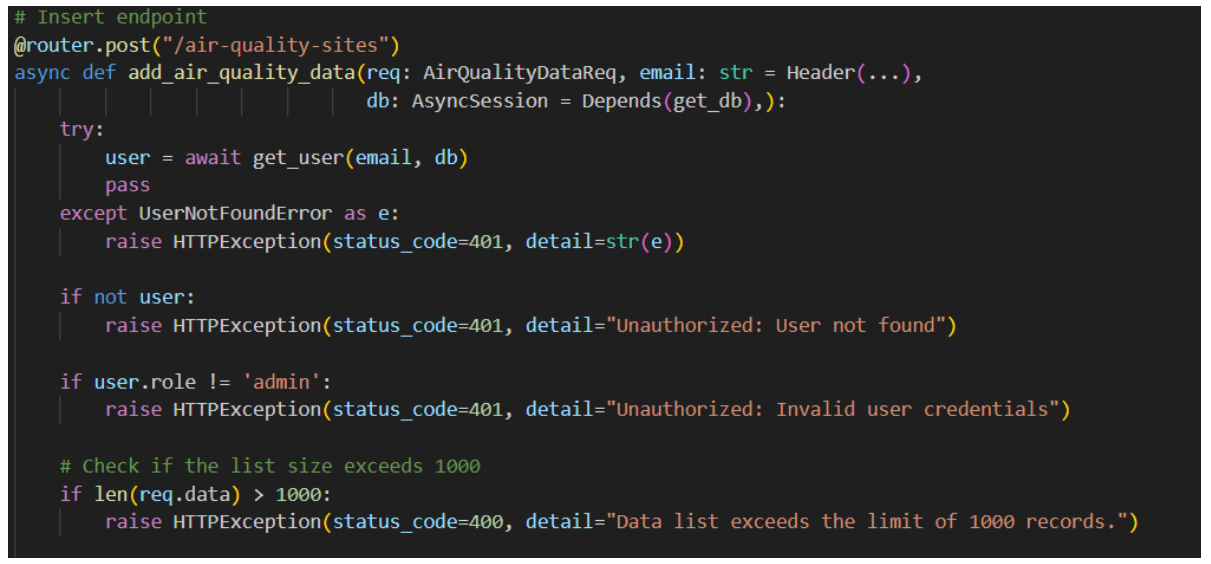


```python
@router.get("/predict")
async def predict(
    code: Literal["GB", "KNN", "LR", "NN", "RF"] = Query(..., description="Model code"),
    from_date: Optional[datetime] = Query(None, description="From date (YYYY-MM-DD)"),
    to_date: Optional[datetime] = Query(None, description="To date (YYYY-MM-DD)"),
    db: AsyncSession = Depends(get_db)
):
    # if code == "NN":
    #     raise HTTPException(status_code=501, detail="NN model temporarily unavailable.")

    # Validate date range
    if from_date and to_date:
        if from_date > to_date:
            raise HTTPException(status_code=400, detail="from_date must be before to_date")

    model_filename = MODEL_MAP.get(code.upper())
    if not os.path.exists(f"models/{model_filename}"):
        raise HTTPException(status_code=500, detail="Model file not found")

    try:
        result = await predict_with_model(f"models/{model_filename}", db, from_date, to_date
        return {
            "model": code,
            "date_range": {
                "from": from_date.isoformat() if from_date else None,
                "to": to_date.isoformat() if to_date else None
            },
            # "count": len(predictions),
            "data": result
        }
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))
```

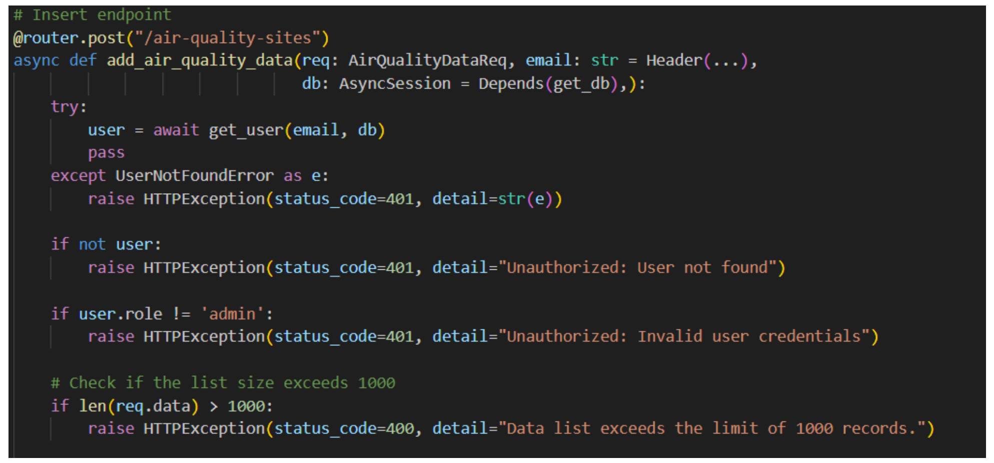


```python
# Insert endpoint
@router.post("/air-quality-sites")
async def add_air_quality_data(req: AirQualityDataReq, email: str = Header(...),
                               db: AsyncSession = Depends(get_db),):
    try:
        user = await get_user(email, db)
        pass
    except UserNotFoundError as e:
        raise HTTPException(status_code=401, detail=str(e))

    if not user:
        raise HTTPException(status_code=401, detail="Unauthorized: User not found")

    if user.role != 'admin':
        raise HTTPException(status_code=401, detail="Unauthorized: Invalid user credentials")

    # Check if the list size exceeds 1000
    if len(req.data) > 1000:
        raise HTTPException(status_code=400, detail="Data list exceeds the limit of 1000 records.")
```

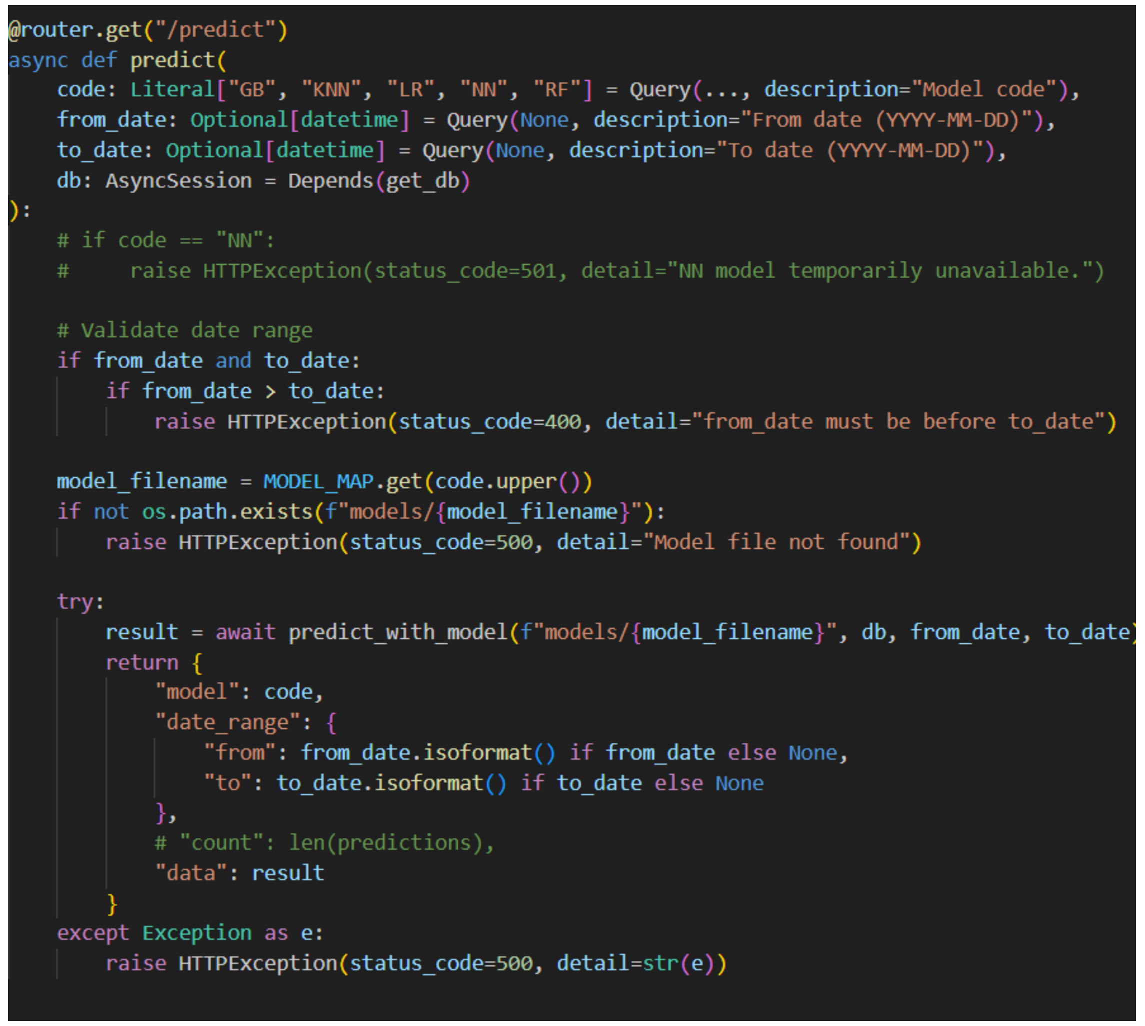


```python
# Insert endpoint
@router.post("/air-quality-sites")
async def add_air_quality_data(req: AirQualityDataReq, email: str = Header(...),
                               db: AsyncSession = Depends(get_db),):
    try:
        user = await get_user(email, db)
        pass
    except UserNotFoundError as e:
        raise HTTPException(status_code=401, detail=str(e))

    if not user:
        raise HTTPException(status_code=401, detail="Unauthorized: User not found")

    if user.role != 'admin':
        raise HTTPException(status_code=401, detail="Unauthorized: Invalid user credentials")

    # Check if the list size exceeds 1000
    if len(req.data) > 1000:
        raise HTTPException(status_code=400, detail="Data list exceeds the limit of 1000 records.")
```



```python
@router.get("/predict")
async def predict(
    code: Literal["GB", "KNN", "LR", "NN", "RF"] = Query(..., description="Model code"),
    from_date: Optional[datetime] = Query(None, description="From date (YYYY-MM-DD)"),
    to_date: Optional[datetime] = Query(None, description="To date (YYYY-MM-DD)"),
    db: AsyncSession = Depends(get_db)
):
    # if code == "NN":
    #     raise HTTPException(status_code=501, detail="NN model temporarily unavailable.")

    # Validate date range
    if from_date and to_date:
        if from_date > to_date:
            raise HTTPException(status_code=400, detail="from_date must be before to_date")

    model_filename = MODEL_MAP.get(code.upper())
    if not os.path.exists(f"models/{model_filename}"):
        raise HTTPException(status_code=500, detail="Model file not found")

    try:
        result = await predict_with_model(f"models/{model_filename}", db, from_date, to_date
        return {
            "model": code,
            "date_range": {
                "from": from_date.isoformat() if from_date else None,
                "to": to_date.isoformat() if to_date else None
            },
            # "count": len(predictions),
            "data": result
        }
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))
```

MODELS USED FOR GENERATING CALBIRATED VALUES

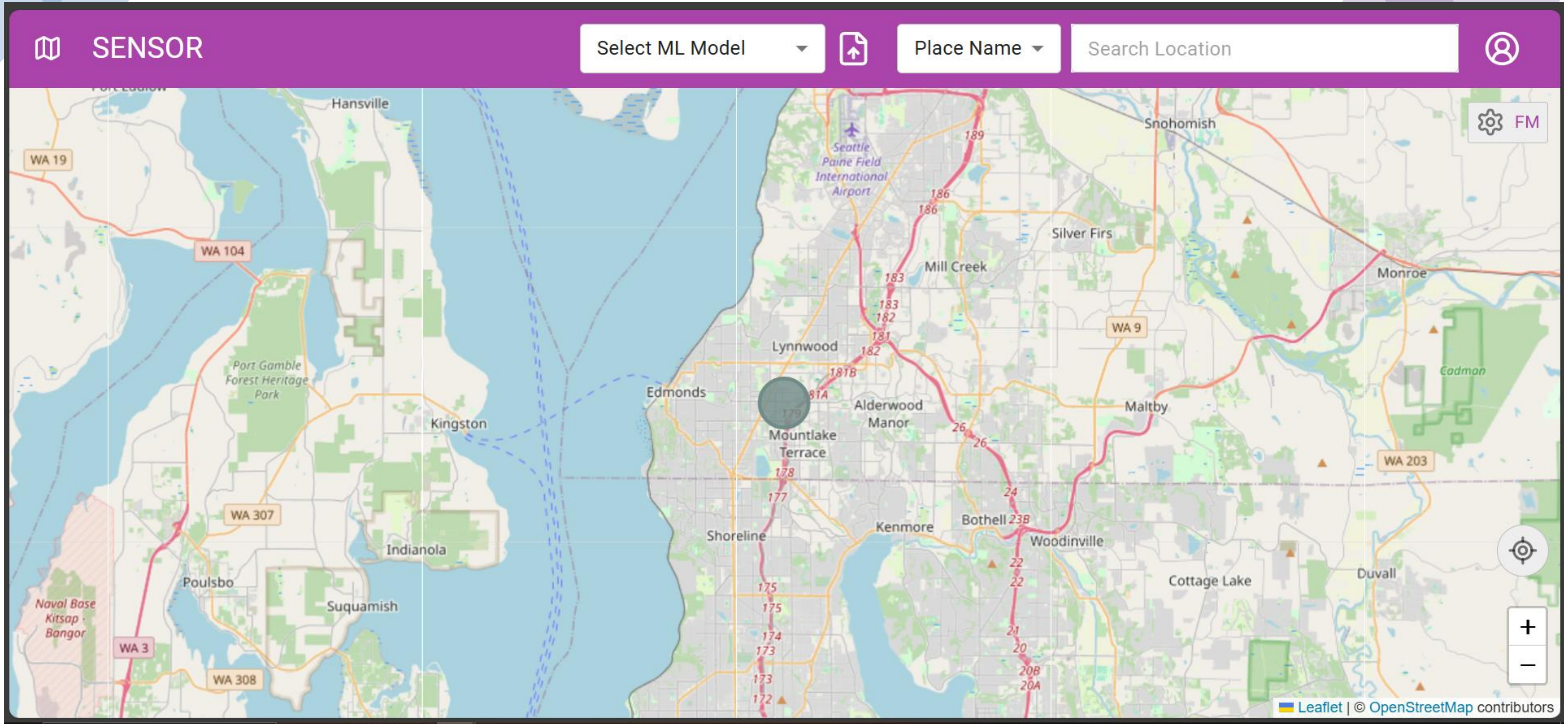
- RANDOM FOREST
- NEURAL NETWORKS
- LOGISTIC REGRESSION
- GRADIENT BOOSTING
- K-NEAREST NEIGHBORS

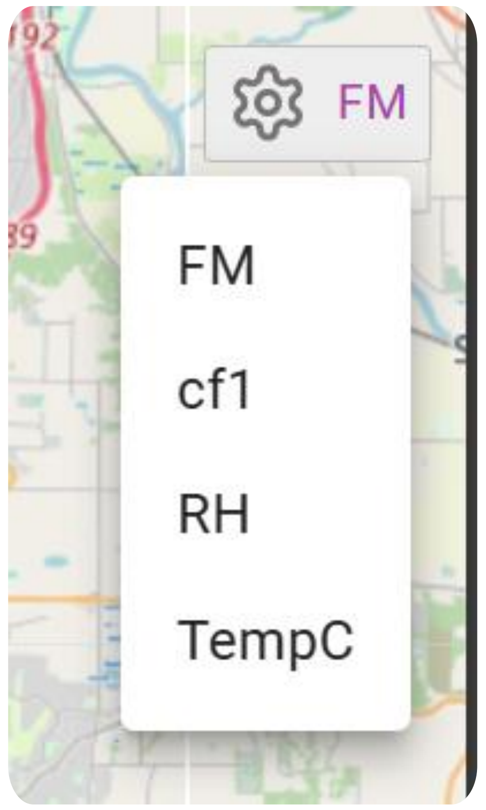
DEPOLYMENT OF THE WEBSITE

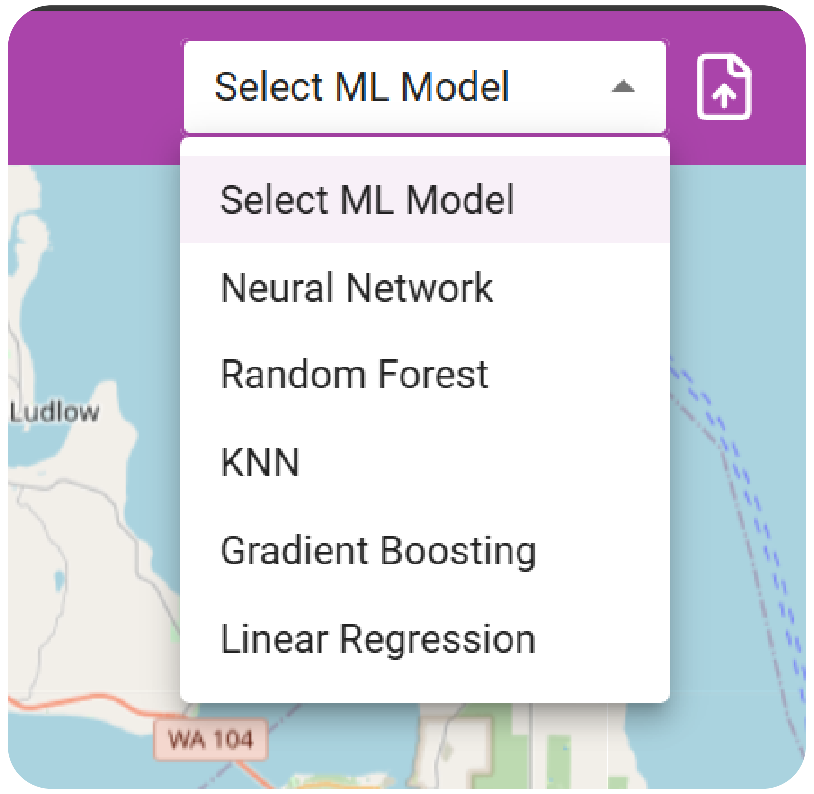
BACKEND: DOCKER CONTAINERS ON AWS EC2.

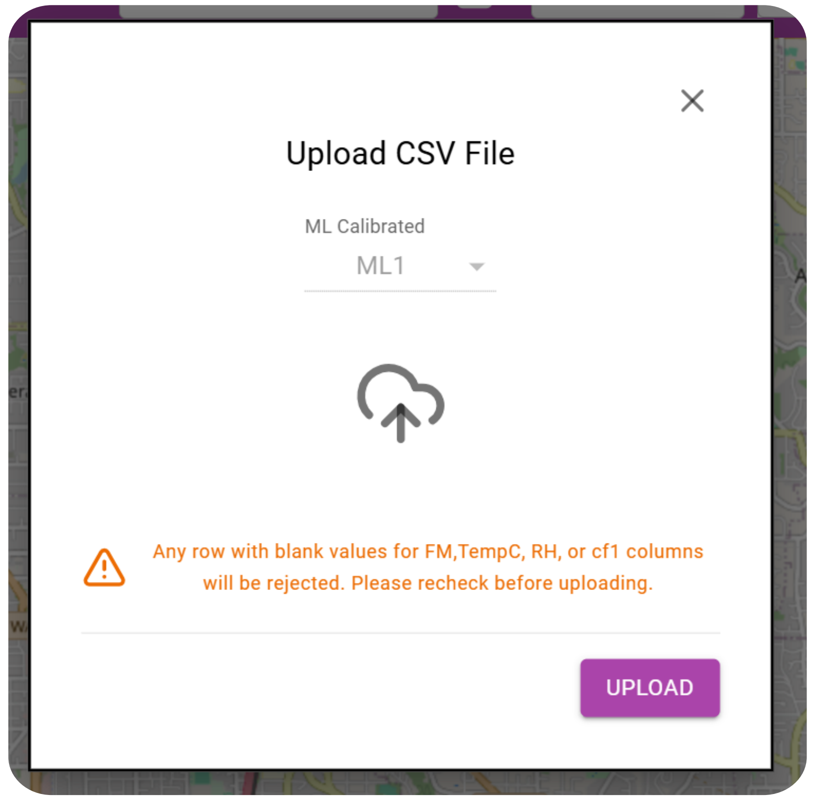
FRONTEND: CLOUDFLARE PAGES.

CLOUDFLARE WORKER: REVERSE PROXY FOR SECURE API ACCESS.


SENSOR
Select ML Model
Place Name
Search Location
FM
Hansville
Port Gamble Forest Heritage Park
Kingston
Indianola
Suquamish
Poulsbo
Naval Base Kitsap - Bangor
Edmonds
Lynnwood
Mountlake Terrace
Alderwood Manor
Shoreline
Kenmore
Bothell
Woodinville
Mill Creek
Silver Firs
Snohomish
Monroe
Maltby
Cottage Lake
Duvall
Seattle Paine Field International Airport
Leaflet | © OpenStreetMap contributors






Upload CSV File

ML Calibrated
ML1

Any row with blank values for FM,TempC, RH, or cf1 columns will be rejected. Please recheck before uploading.

UPLOAD

USER FUNCTIONALITIES

- When users visit the website, they are presented with main user functionalities.

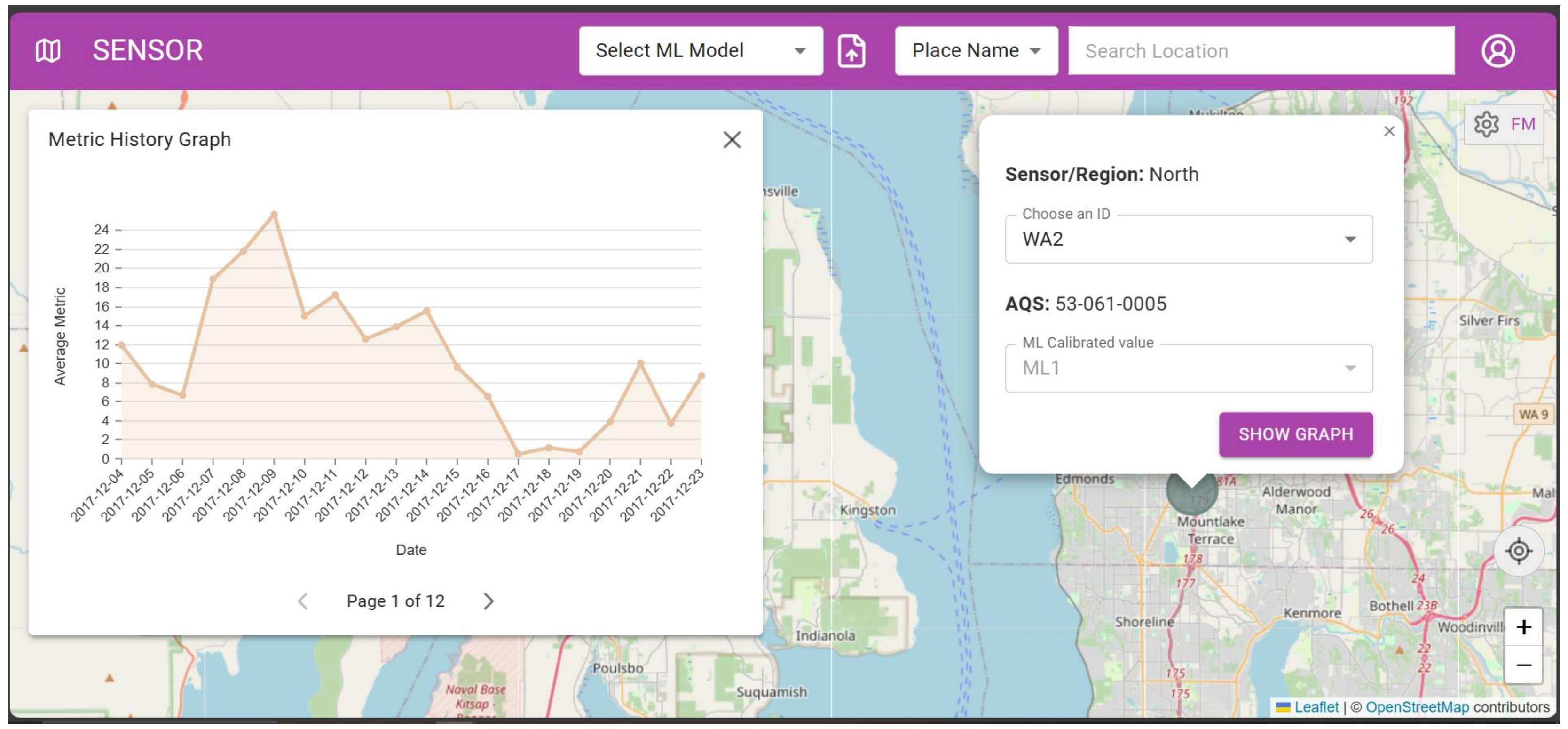
1.SEARCH BAR

2.UPLOADING DATA SET

3.ML MODEL SLECTION

4.SENSOR FILTER

AQI GRAPH GENERATION



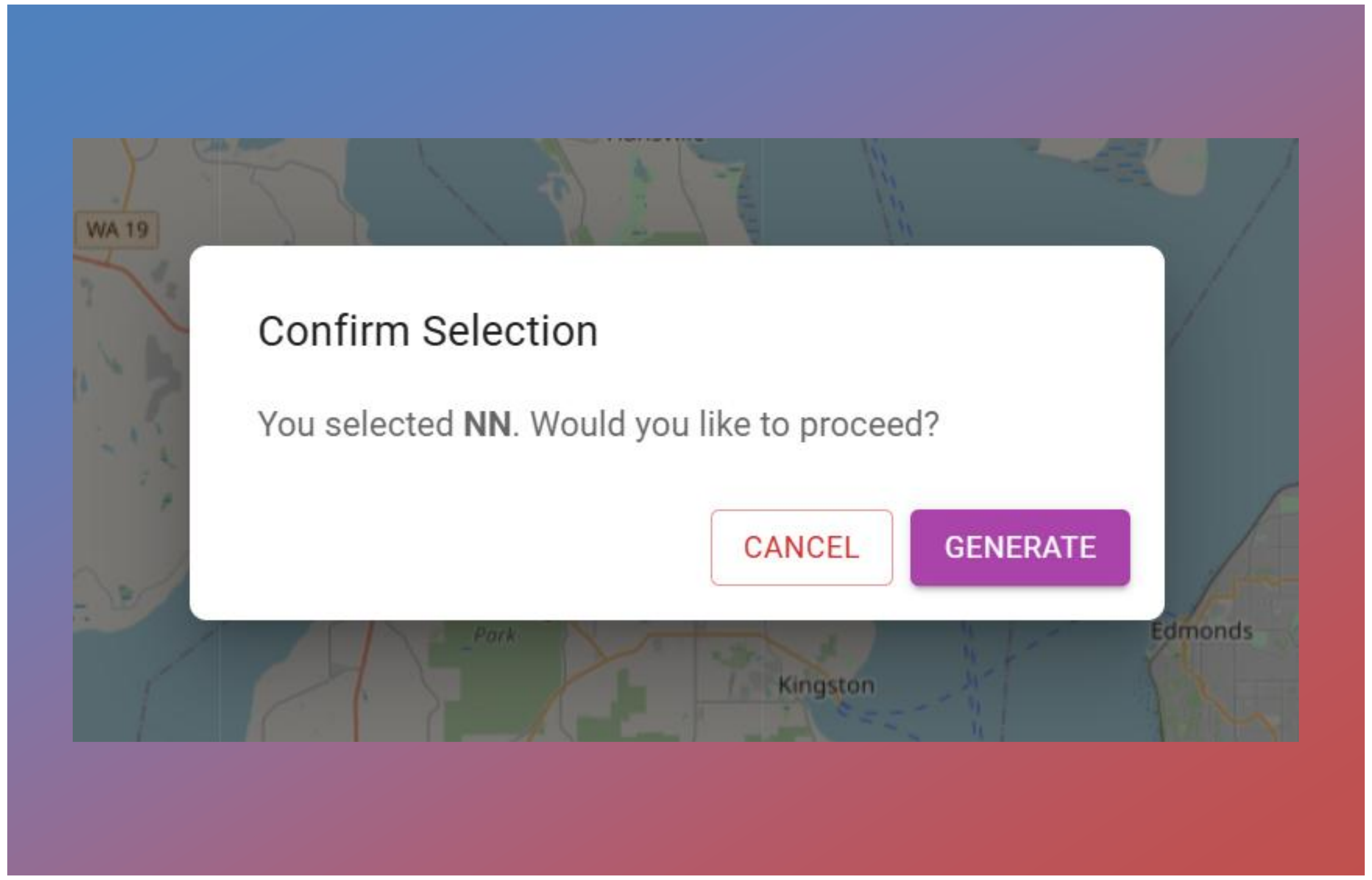
Sensor/Region: North

Choose an ID: WA2

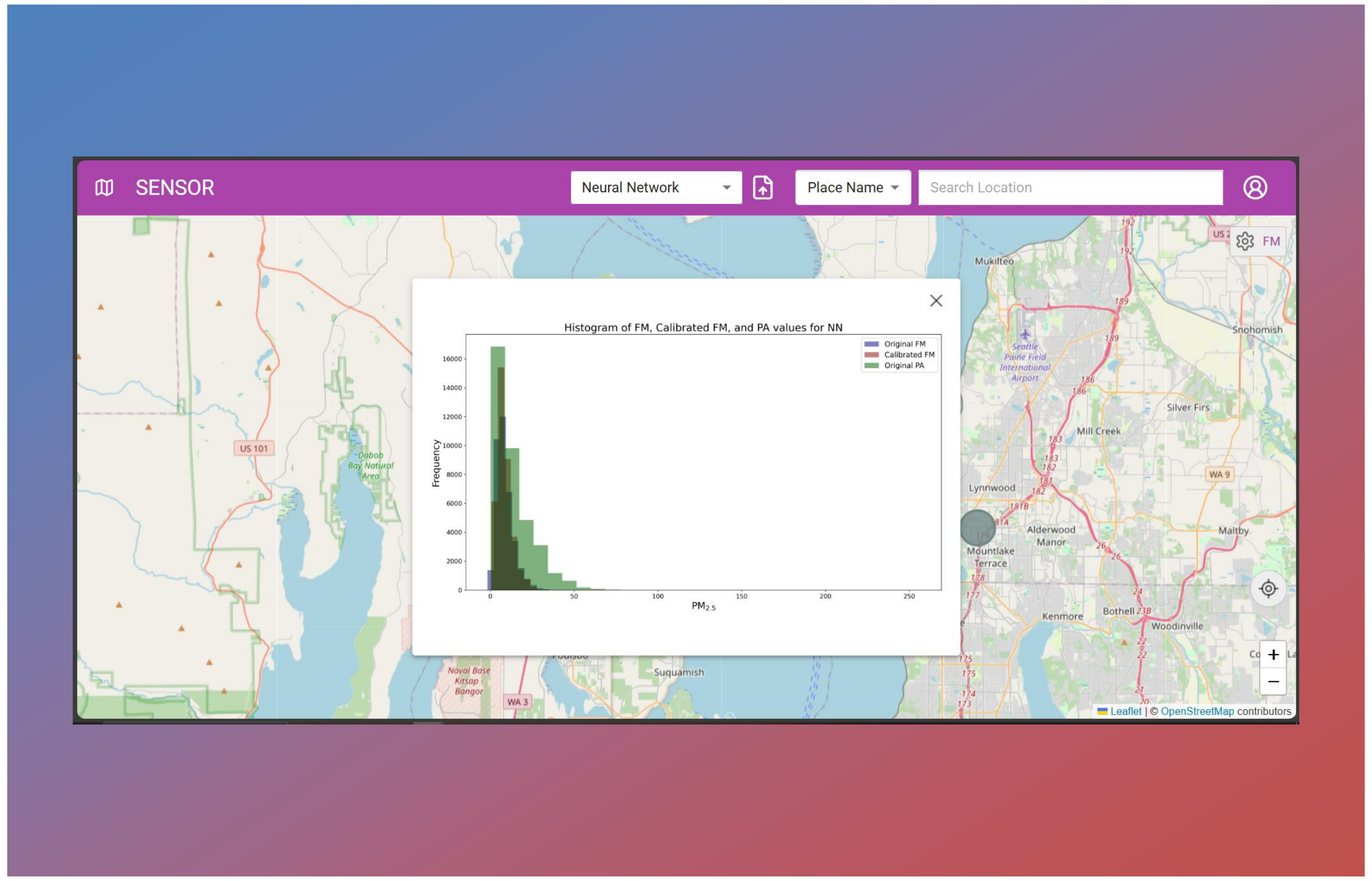
AQS: 53-061-0005

ML Calibrated value: ML1

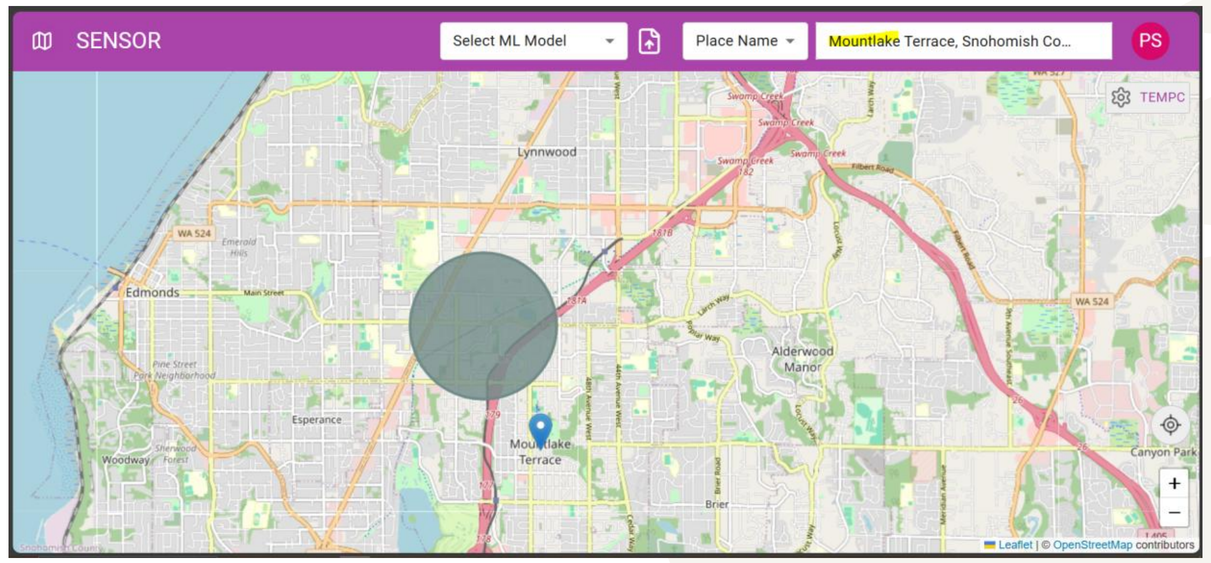
- The dashboard features an interactive map where users can select regions or specific sensor IDs to view detailed metrics.
- Selecting a sensor (such as Filter FM) automatically generates a graph for that sensor, with the option to choose different sensor IDs for more granular analysis.
- The generated graphs display all relevant data for the selected sensor within the specified date range, enabling comprehensive trend analysis and comparison.
- The search bar allows users to quickly find sensor locations by entering a region name (e.g., "Alaska"); the map centers and highlights the exact sensor location, enabling fast and precise navigation.

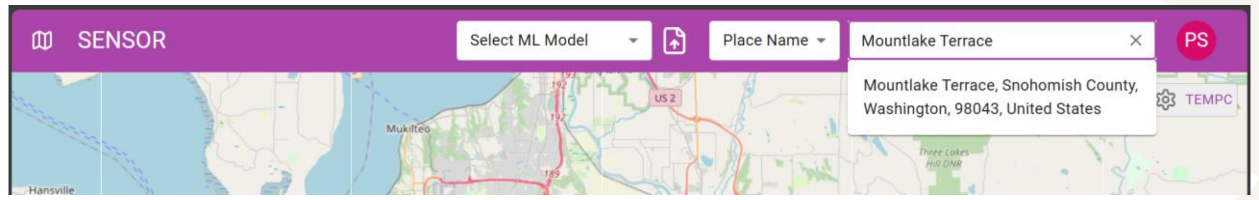

Confirm Selection

You selected **NN**. Would you like to proceed?

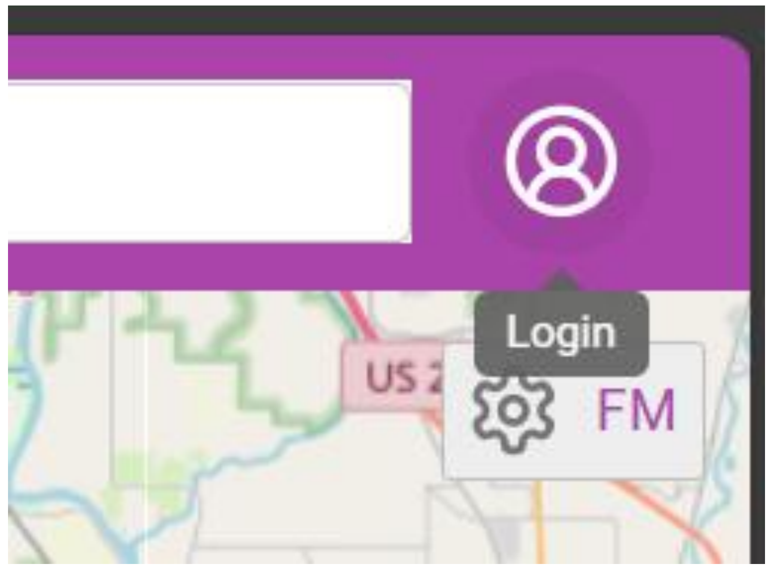


- The platform allows users to choose from multiple machine learning models, which then generate histograms comparing original and calibrated values.
- The user can select any 5 models
- Upon model selection, a confirmation popup appears (e.g., “You selected NN, would you like to proceed?”).
- When the user confirms, the chosen model runs in the background and generates a histogram comparing original and calibrated values, which is displayed as a downloadable image.





Mountlake Terrace, Snohomish County, Washington, 98043, United States



OTHER FUNCTIONALITIES

Users can efficiently search for locations on the map using the autocomplete suggestion feature, which provides instant place suggestions as they type.

Before logging in, users are prompted to sign in with their Google account, ensuring secure access to the platform; after successful authentication, the full dashboard functionality becomes available, including data upload and visualization tools.

When a location is selected from the autocomplete suggestions, a pointer or marker appears on the map at the corresponding coordinates, visually highlighting the searched location for easy reference.

RESULT

- The system successfully delivers accurate AQI predictions across multiple sensor inputs. Model comparison shows Random Forest yielding the best performance with RMSE of 12.3.
- Histograms and line plots help users interpret predictions spatially