RIVER POLLUTANT VISUALIZATION SOFTWARE

Ahmet Furkan Kavraz

Introduction

As industrialization increases to meet human needs, the environment that requires our protection is being increasingly harmed. The discharge of waste from factories into seas and rivers contributes to this issue. In response, the Sustainable Watershed Management Through IoT-Driven Artificial Intelligence (SWAIN) project [1] is developing a system that monitors pollution levels in rivers and notifies authorities to take necessary precautions. My project is a part of this initiative and involves visualizing and presenting the collected pollution data to users.

System Model

Our application development involves three phases:

- Frontend Application
- Backend Application
- Database

Frontend Application: Users can use the application to monitor the pollution in rivers, changes in observed values, and make the minor changes in measurement data.

Backend Application: Devices or other applications, such as AI prediction models, can send requests to make changes in river, device, or measurements data using scripts that we provided.

Database: Database can be only reachable via backend application and we prefer MongoDB, since we use geo-data.

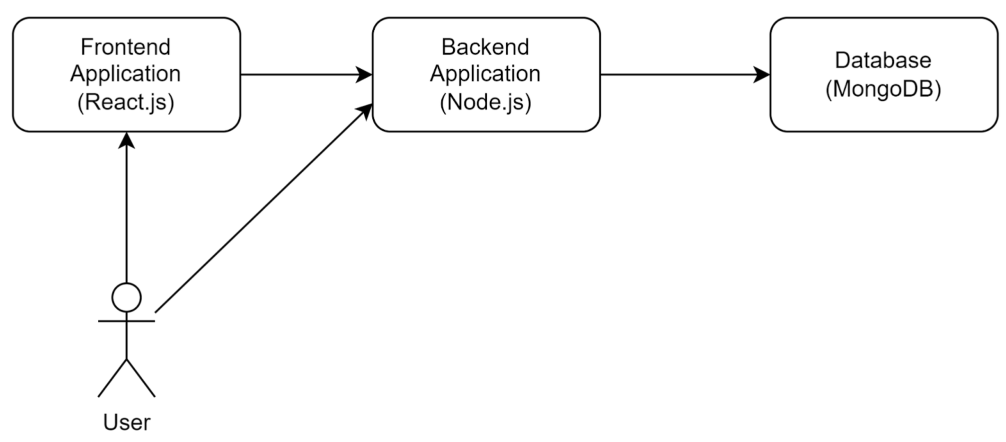

Figure 1: System Design of the Application

Data Model

The data model is an adapted version of real-life in our software. Similar to the real world, rivers are composed of river branches, and these branches are further made up of river points. Some of these points may contain our measurement devices. You can see the diagrams of the model we described in Figure 2 and 3.

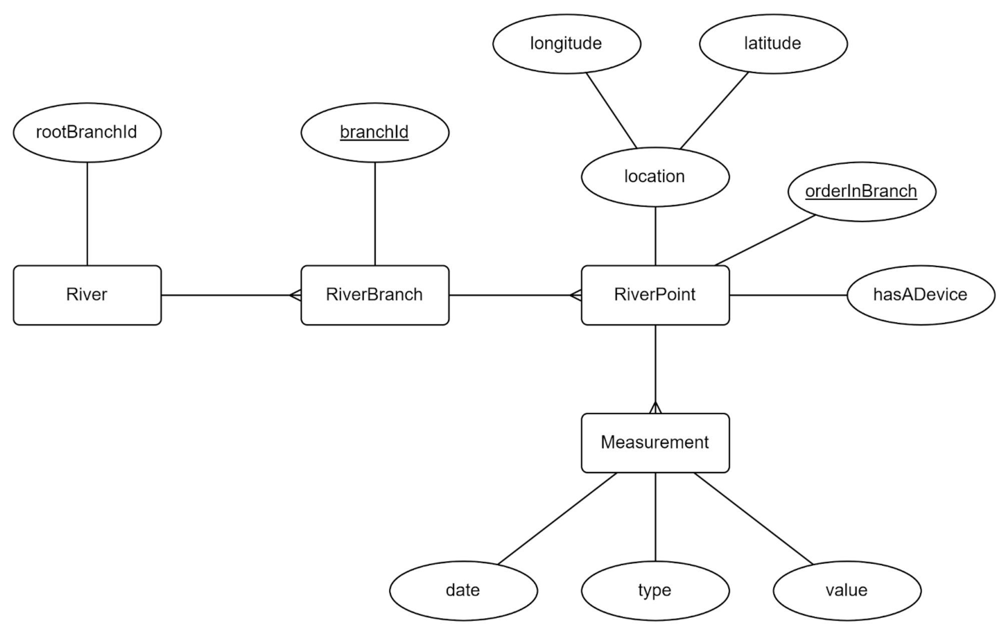

Figure 2: E-R Diagram of the System

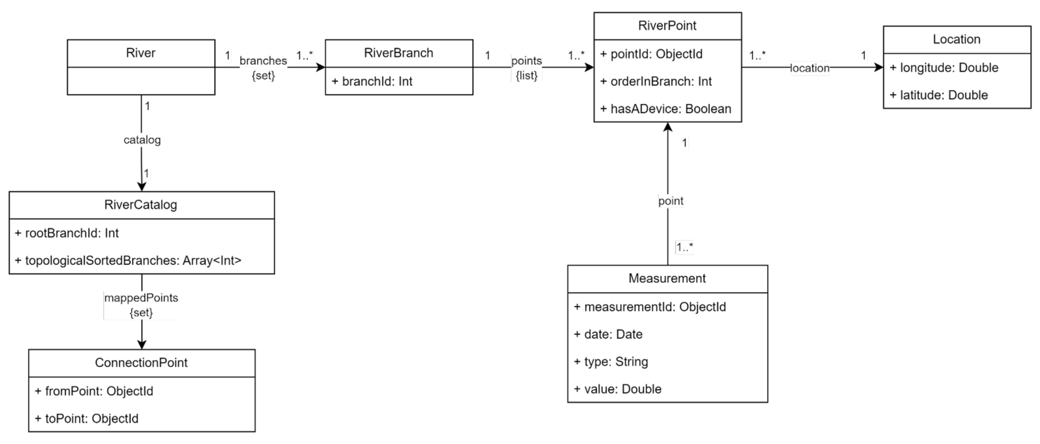

Figure 3: Class Diagram of the System

Algorithms

This section mainly aims to adapt the data schema that is provided by customer and interpolate the measurements for entire river. There are 3 algorithms that are proposed, and they are designed as flexible algorithms.

Connect the River Branches

Figure 4 shows how river data is represented, and the algorithm in Figure 5 is used to connect the river branches and find crossing points.

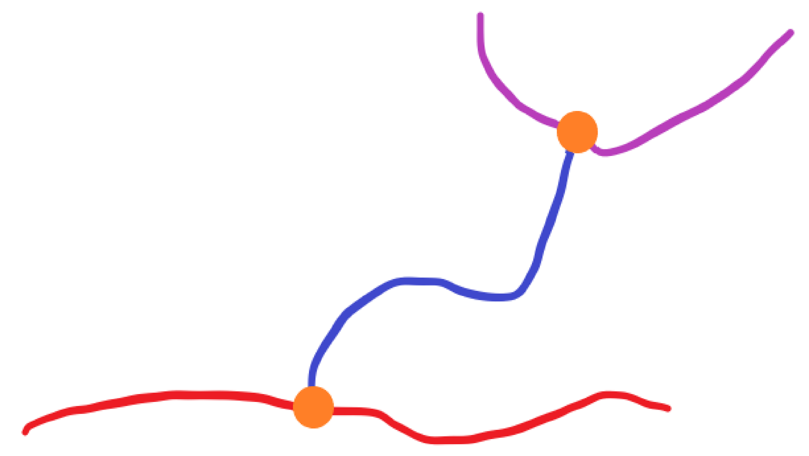

Figure 4: Example Illustration of River Branches

```
Algorithm 1 FindCrossingPointsAndTopologicalSort
Data: river, threshold, rootBranchId
Result: crossingPoints, topologicalSortedArray
crossingPoints ← empty list
adjacencyList ← empty undirected graph
foreach branch ∈ river do
    startingPoint ← findStartingPoint(branch)
    finishingPoint ← findFinishingPoint(branch)
    closestPoint1 ← findClosestPoint(river, startingPoint)
    closestPoint2 ← findClosestPoint(river, finishingPoint)
    if closestPoint1.branchId ≠ branch.branchId then
        if distance(startingPoint, closestPoint1) < threshold then
            crossingPoints.append((branch, closestPoint1))
            adjacencyList.addEdge(branch.branchId, closestPoint1.branchId)
            adjacencyList.addEdge(closestPoint1.branchId, branch.branchId)
        end
    end
    if closestPoint2.branchId ≠ branch.branchId then
        if distance(finishingPoint, closestPoint2) < threshold then
            crossingPoints.append((branch, closestPoint2))
            adjacencyList.addEdge(branch.branchId, closestPoint2.branchId)
            adjacencyList.addEdge(closestPoint2.branchId, branch.branchId)
        end
    end
end
topologicalSortedArray ← dfs(adjacencyList, rootBranchId)
return crossingPoints, topologicalSortedArray
```

Figure 5: Pseudocode of the Algorithm for finding Crossing Points and Topologically Sorted Branches

Interpolation for a Branch

Figure 6 shows how a river branch is represented in our system, and the interpolation algorithm is shown in Figure 7.

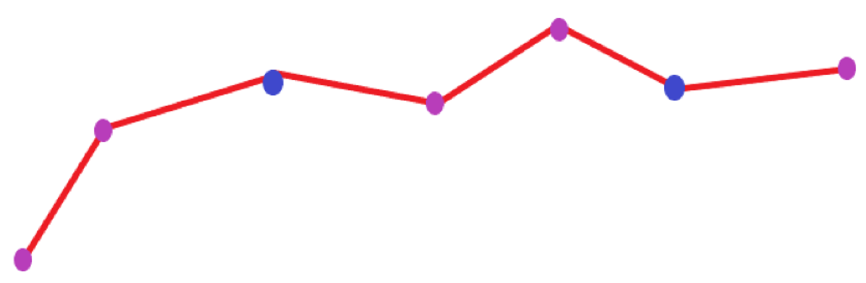

Figure 6: An Example River Branch

```
Algorithm 2 CalculateInterpolatedValues
Data: branch, measurements
Result: interpolatedValues
interpolatedValues ← empty list
sortMeasurementsByOrderInBranch(measurements)
foreach point ∈ branch do
    closestMeasurements ← findClosestMeasurements(measurements, point)
    if closestMeasurements.size() == 2 then
        measurement1 ← closestMeasurements[0]
        measurement2 ← closestMeasurements[1]
        interpolatedValue ← calculateWeightedAverageByOrderInBranch(point,
          measurement1, measurement2)
        interpolatedValues.append(interpolatedValue)
    end
    else
        closestMeasurement ← closestMeasurements[0]
        interpolatedValues.append(closestMeasurement.value)
    end
end
return interpolatedValues
```

Figure 7: Interpolation Algorithm within a Branch

Interpolation Algorithm

The general interpolation algorithm is represented in Figure 8. The algorithm runs for every interpolation requests. Previous two algorithms are also used within this interpolation algorithm, then, the output is visualized using Google Maps API [2].

```
Algorithm 3 Interpolate
Data: date, type, scaleArray
Result: segmentArray
deviceLocations ← ∅; measurementsMap ← ∅;
Find all measurements for date; foreach measurement do
    if measurement.type = type then
        Append measurement to deviceLocations; Map measurement.value to
          measurementsMap[measurement.pointId][measurement.type];
    end
end
branchIds ← unique branchIds associated with points in deviceLocations;
pointMap ← ∅; measurementsByBranches ← ∅;
Clip points with measurements to create a clipped river map;
Fill pointMap and measurementsByBranches using the clipped points;
Retrieve river catalog with rootBranchId, topologicallySortedBranches, and
  crossing points;
foreach branch in topologicallySortedBranches with measurements do
    foreach point in branch do
        Calculate interpolated measurement values for point; Determine scale
          values for each point;
    end
end
Group segments with the same scale value;
return resulting segment array;
```

Figure 8: Interpolation Algorithm for a River

Application

We completed four pages for monitoring and modifying river data. The application's main focus is on interpolation and serving the results to customers. Photos of the application can be found in Figures 9, 10, 11, and 12.

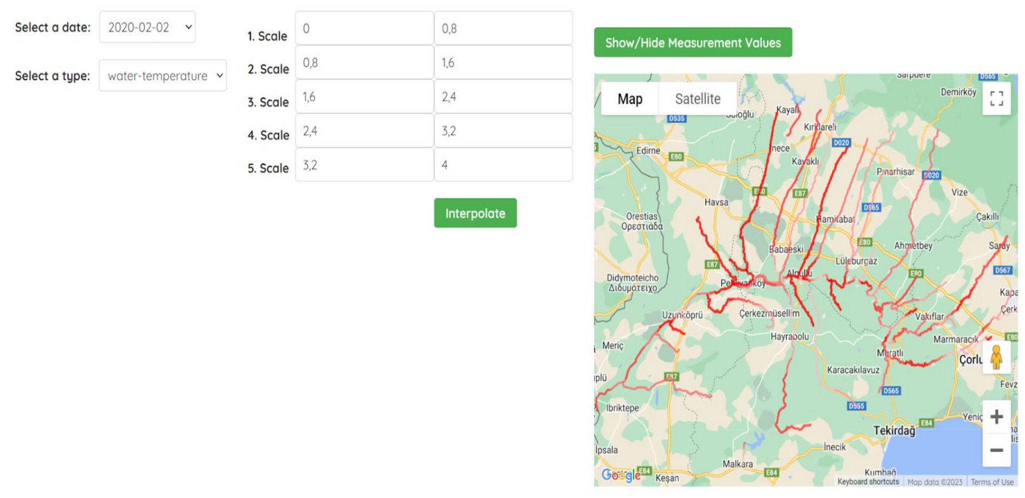

Figure 9: Interpolation Page of the Application

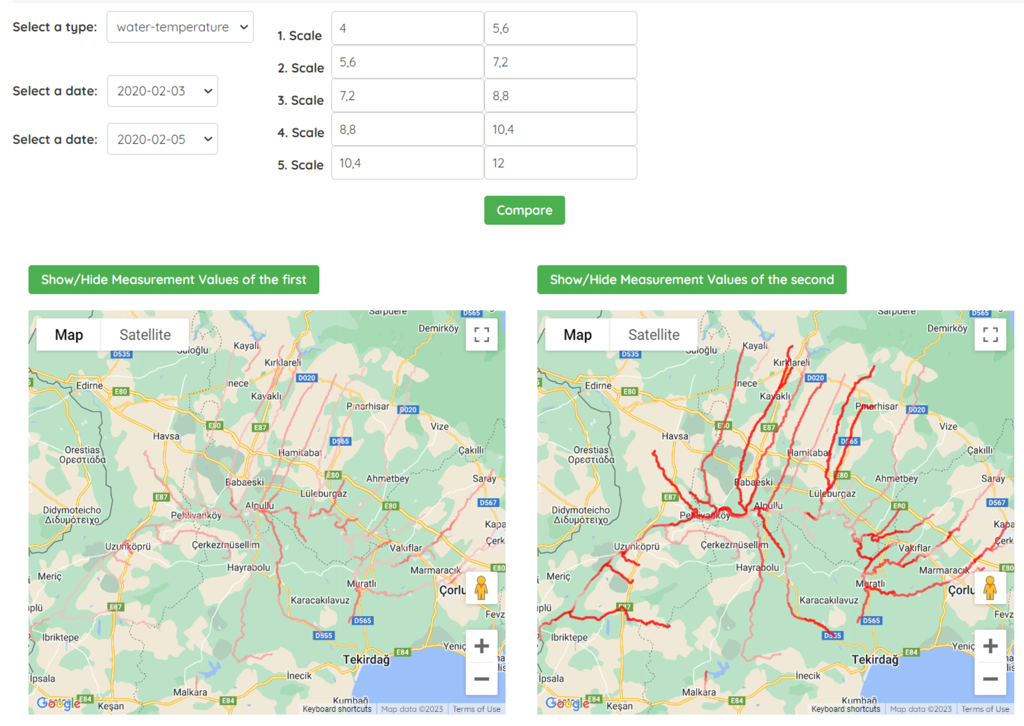

Figure 10: Compare Page of the Application

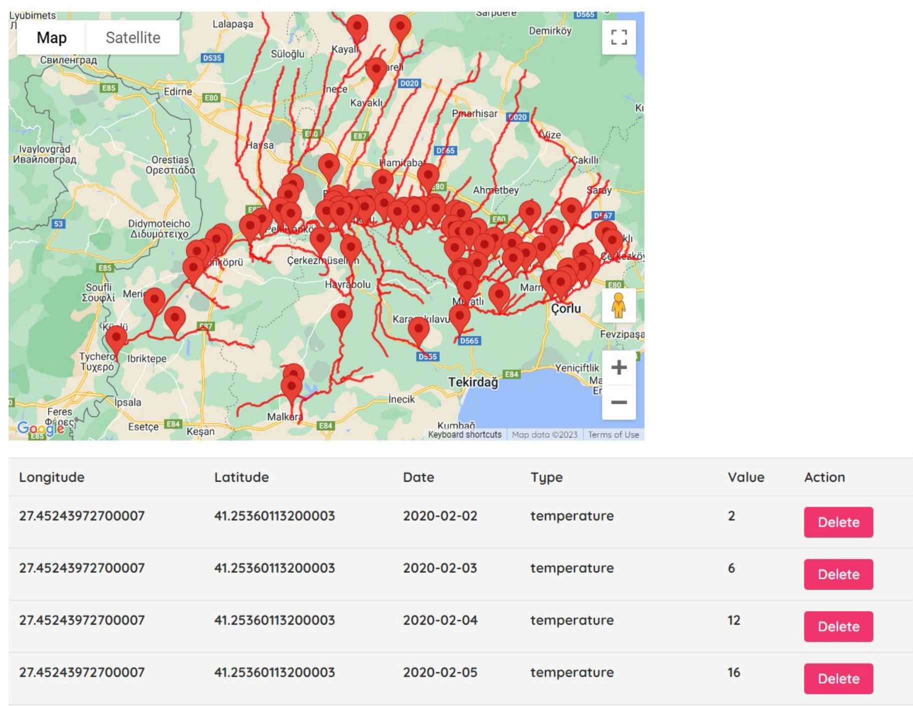

Figure 11: List Measurements Page of the Application

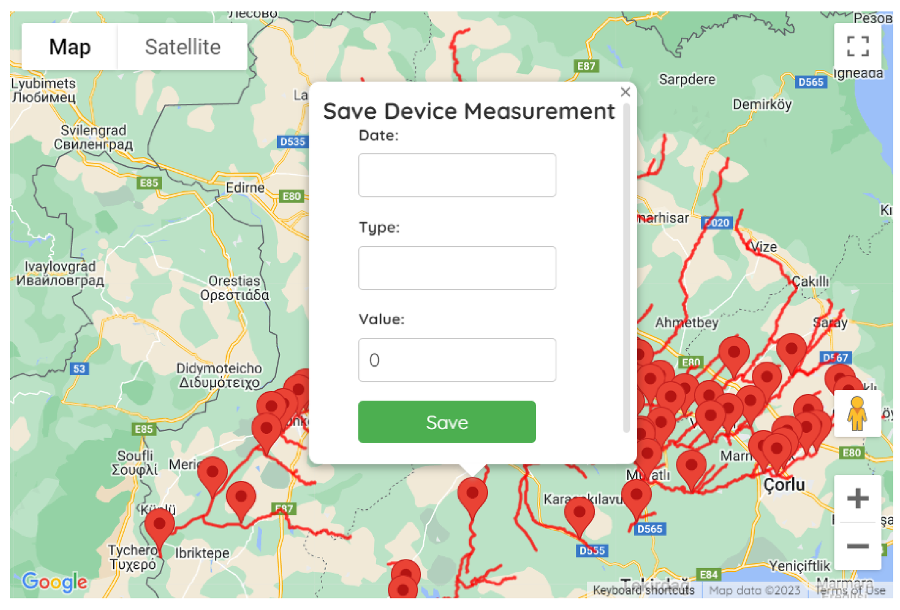

Figure 12: Save Measurements Page of the Application

Evaluation

Key metrics of our application:

- Interpolation requests within 1.2 seconds.
- River creation within 14 seconds.
- River deletion within 1 seconds.
- Device/Measurement operations within 50 milliseconds.

Conclusion

In conclusion, we completed a visualization application to monitor rivers, and detect if anything suspicious. The software will be used by environmental authorities and authorized institutions as a helper tool. In the future, the software can be used by real-people and can be developed according to their recommendations and needs.

References

1. SWAIN Project, https://swain-project.eu/
2. Google Maps Platform, https://developers.google.com/maps

İTÜ
ISTANBUL TECHNICAL UNIVERSITY
Department of Computer Engineering

Faculty of Computer and Informatics Engineering
İTÜ Ayazaga Campus, 34469, Maslak/Istanbul
bbf@itu.edu.tr
http://bm.itu.edu.tr