# United States Natural Disaster Service

Design Document

Table of Contents

[United States Natural Disaster Service 1](#_Toc191632425)

[Problem Statement 3](#_Toc191632426)

[Use Cases 3](#_Toc191632427)

[U1: Disaster Details 3](#_Toc191632428)

[U2: Topological Map 3](#_Toc191632429)

[U3: Create Disaster 3](#_Toc191632430)

[U4: Update Disaster 3](#_Toc191632431)

[Project Scope 3](#_Toc191632432)

[In Scope 3](#_Toc191632433)

[Out of Scope 4](#_Toc191632434)

[Open Questions 4](#_Toc191632435)

[Functional Requirements 5](#_Toc191632436)

[U1: Disaster Details 5](#_Toc191632437)

[FR 1.1: valid\_inputs\_get\_disaster\_returns\_disaster\_details 5](#_Toc191632438)

[FR 1.2: no\_matching\_disaster\_get\_disaster\_not\_found\_error 5](#_Toc191632439)

[U2: Topological Map 5](#_Toc191632440)

[FR 2.1: current\_disasters\_map\_current\_disasters\_returns\_disasters 5](#_Toc191632441)

[FR 2.2: no\_current\_disasters\_map\_current\_disasters\_returns\_empty\_map 6](#_Toc191632442)

[FR 2.3: click\_on\_disaster\_area\_renders\_popup 6](#_Toc191632443)

[U3: Create Disaster 6](#_Toc191632444)

[FR 3.1: valid\_administrator\_create\_disaster\_creates\_disaster 6](#_Toc191632445)

[FR 3.2: unauthorized\_user\_create\_disaster\_forbidden\_error 7](#_Toc191632446)

[FR 3.3: missing\_data\_create\_disaster\_bad\_request\_error 7](#_Toc191632447)

[FR 3.4: future\_start\_date\_create\_disaster\_bad\_request\_error 7](#_Toc191632448)

[FR 3.5: invalid\_disaster\_type\_create\_disaster\_bad\_request\_error 8](#_Toc191632449)

[FR 3.6: invalid\_severity\_create\_disaster\_bad\_request\_error 8](#_Toc191632450)

[FR 3.7 negative\_deaths\_create\_disaster\_bad\_request\_error 8](#_Toc191632451)

[FR 3.8: negative\_injuries\_create\_disaster\_bad\_request\_error 8](#_Toc191632452)

[FR 3.9: negative\_financial\_impact\_create\_disaster\_bad\_request\_error 8](#_Toc191632453)

[FR 3.10: too\_many\_decimals\_financial\_impact\_create\_disaster\_bad\_request\_error 9](#_Toc191632454)

[FR 3.11: invalid\_location\_create\_disaster\_bad\_request\_error 9](#_Toc191632455)

[FR 3.12: invalid\_area\_create\_disaster\_bad\_request\_error 9](#_Toc191632456)

[U4: Update Disaster 9](#_Toc191632457)

[FR 4.1: valid\_administrator\_update\_disaster\_creates\_disaster 9](#_Toc191632458)

[FR 4.2: unauthorized\_user\_update\_disaster\_forbidden\_error 10](#_Toc191632459)

[FR 4.3: disaster\_does\_not\_exist\_update\_disaster\_not\_found\_error 10](#_Toc191632460)

[FR 4.4: missing\_data\_update\_disaster\_bad\_request\_error 10](#_Toc191632461)

[FR 4.5: future\_start\_date\_update\_disaster\_bad\_request\_error 11](#_Toc191632462)

[FR 4.6: future\_end\_date\_update\_disaster\_bad\_request\_error 11](#_Toc191632463)

[FR 4.7: invalid\_disaster\_type\_update\_disaster\_bad\_request\_error 11](#_Toc191632464)

[FR 4.8: invalid\_severity\_update\_disaster\_bad\_request\_error 11](#_Toc191632465)

[FR 4.9: negative\_deaths\_update\_disaster\_bad\_request\_error 11](#_Toc191632466)

[FR 4.10: negative\_injuries\_update\_disaster\_bad\_request\_error 12](#_Toc191632467)

[FR 4.11: negative\_financial\_impact\_update\_disaster\_bad\_request\_error 12](#_Toc191632468)

[FR 4.12: too\_many\_decimals\_financial\_impact\_update\_disaster\_bad\_request\_error 12](#_Toc191632469)

[FR 4.13: invalid\_location\_update\_disaster\_bad\_request\_error 12](#_Toc191632470)

[FR 4.14: invalid\_area\_update\_disaster\_bad\_request\_error 13](#_Toc191632471)

[UX Design 13](#_Toc191632472)

[Mockups 13](#_Toc191632473)

[Map Page 13](#_Toc191632474)

[Popup 13](#_Toc191632475)

[Details Page 14](#_Toc191632476)

[Proof Of Concept 14](#_Toc191632477)

[Map Page 14](#_Toc191632478)

[Popup 14](#_Toc191632479)

[Proposed Architecture 15](#_Toc191632480)

[High-Level Overview 15](#_Toc191632481)

[Architectural Overview 15](#_Toc191632482)

[Components 15](#_Toc191632483)

[Chosen Technologies 16](#_Toc191632484)

[Service Design 17](#_Toc191632485)

[Disaster Backend Service 17](#_Toc191632486)

[Internal Models 18](#_Toc191632487)

[Components 18](#_Toc191632488)

[API Design 18](#_Toc191632489)

[Data Design 22](#_Toc191632490)

[Table 1: Disasters 22](#_Toc191632491)

[Development Operations 22](#_Toc191632492)

[Code Review Process 23](#_Toc191632493)

[Deployment Process 23](#_Toc191632494)

[Logging, Monitoring, & Metrics 23](#_Toc191632495)

[Tasks and Milestones 23](#_Toc191632496)

[Milestone 1: <Title> 23](#_Toc191632497)

[Task 1.1: <Title> 23](#_Toc191632498)

[Appendix 1: Definition of Terms 23](#_Toc191632499)

[Appendix 2: Citations 24](#_Toc191632500)

[Appendix 3: Diagram Sources 24](#_Toc191632501)

[Service Design 24](#_Toc191632502)

[Disaster Backend Service 24](#_Toc191632503)

[Data Design 29](#_Toc191632504)

[Table 1: Disasters 29](#_Toc191632505)

# Problem Statement

The United States continues to face a number of natural disasters on a regular basis. The goal of our service is to track the statistics of natural disasters across the United States.

In this design document, we will outline how we plan to implement the United States Natural Disaster Service. This service lets us track natural disasters by disaster name, state, disaster type, danger level, and date.

# Use Cases

## U1: Disaster Details

As a United States Natural Disaster Service customer, I want to get a detailed explanation of a disaster when I open a disaster details page.

## U2: Topological Map

As a United States Natural Disaster Service customer, I want to be able to access a topological map showing ongoing natural disasters.

## U3: Create Disaster

As a United States Natural Disaster Service administrator, I want to be able to create new logs of natural disasters.

## U4: Update Disaster

As a United States Natural Disaster Service administrator, I want to be able to update logs of existing natural disasters.

# Project Scope

## In Scope

* Detailed explanation of a disaster on a disaster details page, including:
  + ID
  + Area
    - Polygon (Points – Lattitude & Longitude)
  + Death toll
  + Injuries
  + Financial Impact
  + Start Date
  + End Date
  + Severity
  + Type of disaster
* Logging the following disaster types:
  + Earth quakes
  + Tornadoes
  + Hurricanes
  + Fires
  + Floods
  + Volcano Eruptions
  + Blizzards
  + Tsunamis
* Topological map showing ongoing natural disasters. Clicking on a disaster will show:
  + Type of disaster
  + Start Date
  + Death toll
  + Injuries
  + Severity
* Administrators are able to create new logs of natural disasters
* Administrators are able to update existing natural disasters

## Out of Scope

* Providing a list of resources for ongoing natural disasters
* User profile creation
* Integration with natural disaster resources, such as the USGS and NWS
* Other disasters – hail, blackouts/brownouts, droughts
* Heat map showing the density of disasters by location
* Disaster routes / forecast
* Deleting natural disaster logs
* Reporting erroneous natural disaster logs / Fact Checking
* Allowing non-administrator users to add natural disasters
* Granular incidents within an overall disaster
* Customization of map colors
* Currency selection
* Time Zone selection
* Admin-specific UI

# Open Questions

1. How can we enforce authentication and authorization on creating & updating disasters?
2. What do we need to consider for internationalization and localization of details?
3. Do we want to add names to the disaster entries?

# Functional Requirements

## U1: [Disaster Details](#_U1:_Disaster_Details)

### FR 1.1: valid\_inputs\_get\_disaster\_returns\_disaster\_details

GIVEN:

* Disaster ID
* Existing disaster with the given disaster ID

WHEN:

* User gets the disaster details

THEN:

* Return valid disaster details, including:
  + ID
  + Death toll
  + Injuries
  + Financial Impact
  + Start Date
  + End Date
  + Severity
  + Type of disaster

### FR 1.2: no\_matching\_disaster\_get\_disaster\_not\_found\_error

GIVEN:

* Disaster ID
* No existing disaster with the given disaster ID

WHEN:

* User gets the disaster details

THEN:

* UX: Return a 404 – Not Found Error
* Backend: Return NotFoundError

## U2: [Topological Map](#_U2:_Topological_Map)

### FR 2.1: current\_disasters\_map\_current\_disasters\_returns\_disasters

GIVEN:

* Existing ongoing disasters

WHEN:

* User gets the topological map of current disasters

THEN:

* UX: Return a map rendering the current disasters
  + Each disaster has its own area shown on the map
* Backend: Return a list of ongoing disasters
  + Disasters filtered to those with no end date
  + For each disaster, return:
    - ID
    - Area
      * Polygon (Points – Lattitude & Longitude)
    - Start Date
    - Severity
    - Type of disaster

### FR 2.2: no\_current\_disasters\_map\_current\_disasters\_returns\_empty\_map

GIVEN:

* No existing ongoing disasters

WHEN:

* User gets the topological map of current disasters

THEN:

* UX: Return a map with no disasters shown
* Backend: Return an empty list

### FR 2.3: click\_on\_disaster\_area\_renders\_popup

GIVEN:

* Topological map of current disasters with current disasters

WHEN:

* User clicks on a disaster’s area

THEN:

* UX: Render pop-up of disaster details on map, including:
  + Start Date
  + Severity
  + Type of disaster
  + Link to disaster details page for the given disaster
    - Type of disaster

## U3: [Create Disaster](#_U3:_Create_Disaster)

### FR 3.1: valid\_administrator\_create\_disaster\_creates\_disaster

GIVEN:

* An authenticated, authorized administrator
* A valid disaster, including:
  + Area
    - Polygon (Points – Lattitude & Longitude)
  + Death toll
  + Injuries
  + Financial Impact
  + Start Date
  + Severity
  + Type of disaster

WHEN:

* User creates a disaster

THEN:

* Disaster is added to data store, including:
  + Generated ID
  + Area
    - Polygon (Points – Lattitude & Longitude)
  + Death toll
  + Injuries
  + Financial Impact
  + Start Date
  + End Date (NULL)
  + Severity : 1-5
  + Type of disaster

### FR 3.2: unauthorized\_user\_create\_disaster\_forbidden\_error

GIVEN:

* An unauthorized user

WHEN:

* User creates a disaster

THEN:

* Return a 403 Forbidden error

### FR 3.3: missing\_data\_create\_disaster\_bad\_request\_error

GIVEN:

* A valid administrator password
* A disaster, missing one or more of the following fields:
  + Area
    - Polygon (Points – Lattitude & Longitude)
  + Death toll
  + Injuries
  + Financial Impact (USD)
  + Start Date
  + Severity
  + Type of disaster

WHEN:

* User creates a disaster

THEN:

* Return a 400 Bad Request error

### FR 3.4: future\_start\_date\_create\_disaster\_bad\_request\_error

GIVEN:

* A valid administrator password
* A disaster, with a start date set for the future

WHEN:

* User creates a disaster

THEN:

* Return a 400 Bad Request error

### FR 3.5: invalid\_disaster\_type\_create\_disaster\_bad\_request\_error

GIVEN:

* A valid administrator password
* A disaster, with a non-supported disaster type

WHEN:

* User creates a disaster

THEN:

* Return a 400 Bad Request error

### FR 3.6: invalid\_severity\_create\_disaster\_bad\_request\_error

GIVEN:

* A valid administrator password
* A disaster, with a severity <= 0 or >5

WHEN:

* User creates a disaster

THEN:

* Return a 400 Bad Request error

### FR 3.7 negative\_deaths\_create\_disaster\_bad\_request\_error

GIVEN:

* A valid administrator password
* A disaster, with a death toll < 0

WHEN:

* User creates a disaster

THEN:

* Return a 400 Bad Request error

### FR 3.8: negative\_injuries\_create\_disaster\_bad\_request\_error

GIVEN:

* A valid administrator password
* A disaster, with injuries < 0

WHEN:

* User creates a disaster

THEN:

* Return a 400 Bad Request error

### FR 3.9: negative\_financial\_impact\_create\_disaster\_bad\_request\_error

GIVEN:

* A valid administrator password
* A disaster, with financial\_impact < 0.00

WHEN:

* User creates a disaster

THEN:

* Return a 400 Bad Request error

### FR 3.10: too\_many\_decimals\_financial\_impact\_create\_disaster\_bad\_request\_error

GIVEN:

* A valid administrator password
* A disaster, with financial\_impact with more than 2 decimal pales

WHEN:

* User creates a disaster

THEN:

* Return a 400 Bad Request error

### FR 3.11: invalid\_location\_create\_disaster\_bad\_request\_error

GIVEN:

* A valid administrator password
* A disaster, with an invalid location

WHEN:

* User creates a disaster

THEN:

* Return a 400 Bad Request error

### FR 3.12: invalid\_area\_create\_disaster\_bad\_request\_error

GIVEN:

* A valid administrator password
* A disaster, with an invalid area

WHEN:

* User creates a disaster

THEN:

* Return a 400 Bad Request error

## U4: [Update Disaster](#_U4:_Update_Disaster)

### FR 4.1: valid\_administrator\_update\_disaster\_creates\_disaster

GIVEN:

* An authenticated, authorized administrator
* A valid disaster, including:
  + ID
  + Area
    - Polygon (Points – Lattitude & Longitude)
  + Death toll
  + Injuries
  + Financial Impact
  + Start Date
  + End Date (optional)
  + Severity
  + Type of disaster
* An existing disaster for the provided ID

WHEN:

* User updates a disaster

THEN:

* Disaster is updated with provided disaster information in datastore
  + ID is not updated

### FR 4.2: unauthorized\_user\_update\_disaster\_forbidden\_error

GIVEN:

* An authorized user

WHEN:

* User updates a disaster

THEN:

* Return a 403 Forbidden error

### FR 4.3: disaster\_does\_not\_exist\_update\_disaster\_not\_found\_error

GIVEN:

* A valid administrator password
* A valid disaster
* No existing disaster with a matching ID

WHEN:

* User updates a disaster

THEN:

* Return a 404 Not Found error

### FR 4.4: missing\_data\_update\_disaster\_bad\_request\_error

GIVEN:

* A valid administrator password
* A disaster, missing one or more of the following fields:
  + ID
  + Area
    - Polygon (Points – Lattitude & Longitude)
  + Death toll
  + Injuries
  + Financial Impact (USD)
  + Start Date
  + Severity
  + Type of disaster
* An existing disaster with a matching ID

WHEN:

* User updates a disaster

THEN:

* Return a 400 Bad Request error

### FR 4.5: future\_start\_date\_update\_disaster\_bad\_request\_error

GIVEN:

* A valid administrator password
* A disaster, with a start date set for the future
* An existing disaster with a matching ID

WHEN:

* User updates a disaster

THEN:

* Return a 400 Bad Request error

### FR 4.6: future\_end\_date\_update\_disaster\_bad\_request\_error

GIVEN:

* A valid administrator password
* A disaster, with an end date set for the future
* An existing disaster with a matching ID

WHEN:

* User updates a disaster

THEN:

* Return a 400 Bad Request error

### FR 4.7: invalid\_disaster\_type\_update\_disaster\_bad\_request\_error

GIVEN:

* A valid administrator password
* A disaster, with a non-supported disaster type
* An existing disaster with a matching ID

WHEN:

* User updates a disaster

THEN:

* Return a 400 Bad Request error

### FR 4.8: invalid\_severity\_update\_disaster\_bad\_request\_error

GIVEN:

* A valid administrator password
* A disaster, with a severity <= 0 or > 5
* An existing disaster with a matching ID

WHEN:

* User updates a disaster

THEN:

* Return a 400 Bad Request error

### FR 4.9: negative\_deaths\_update\_disaster\_bad\_request\_error

GIVEN:

* A valid administrator password
* A disaster, with a death toll < 0
* An existing disaster with a matching ID

WHEN:

* User updates a disaster

THEN:

* Return a 400 Bad Request error

### FR 4.10: negative\_injuries\_update\_disaster\_bad\_request\_error

GIVEN:

* A valid administrator password
* A disaster, with injuries < 0
* An existing disaster with a matching ID

WHEN:

* User updates a disaster

THEN:

* Return a 400 Bad Request error

### FR 4.11: negative\_financial\_impact\_update\_disaster\_bad\_request\_error

GIVEN:

* A valid administrator password
* A disaster, with a financial impact < 0.00
* An existing disaster with a matching ID

WHEN:

* User updates a disaster

THEN:

* Return a 400 Bad Request error

### FR 4.12: too\_many\_decimals\_financial\_impact\_update\_disaster\_bad\_request\_error

GIVEN:

* A valid administrator password
* A disaster, with a financial impact with more than 2 decimals
* An existing disaster with a matching ID

WHEN:

* User updates a disaster

THEN:

* Return a 400 Bad Request error

### FR 4.13: invalid\_location\_update\_disaster\_bad\_request\_error

GIVEN:

* A valid administrator password
* A disaster, with an invalid location
* An existing disaster with a matching ID

WHEN:

* User updates a disaster

THEN:

* Return a 400 Bad Request error

### FR 4.14: invalid\_area\_update\_disaster\_bad\_request\_error

GIVEN:

* A valid administrator password
* A disaster, with an invalid area
* An existing disaster with a matching ID

WHEN:

* User updates a disaster

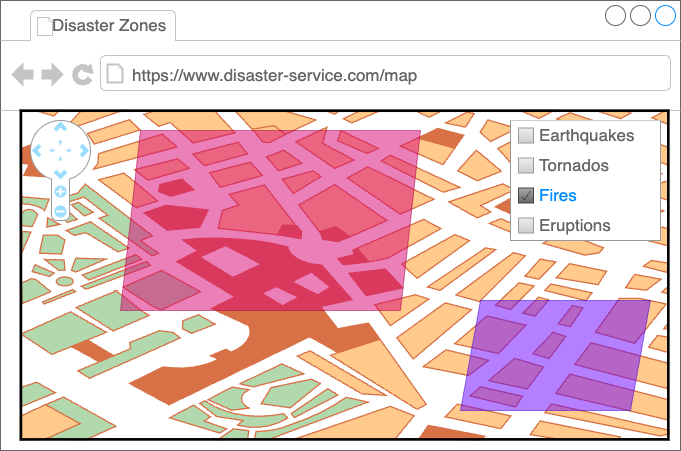
THEN:

* Return a 400 Bad Request error

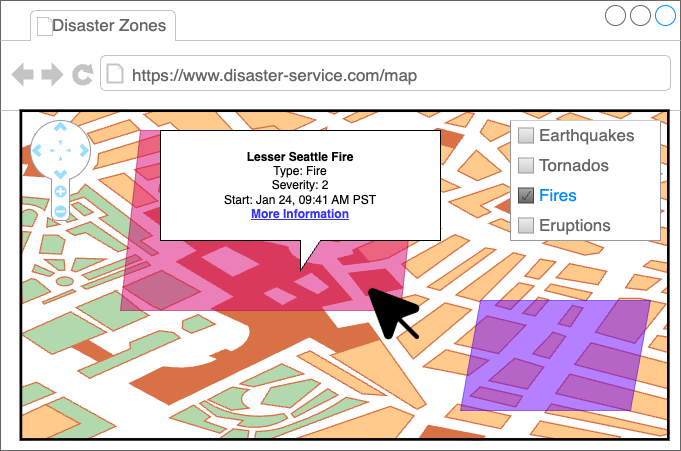
# UX Design

## Mockups

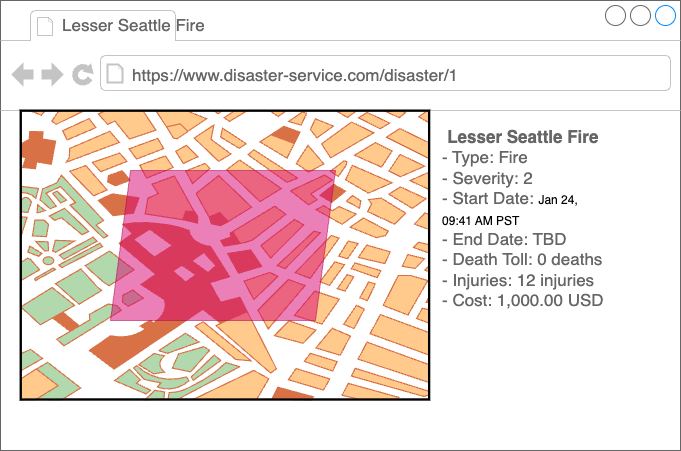
### Map Page



### Popup

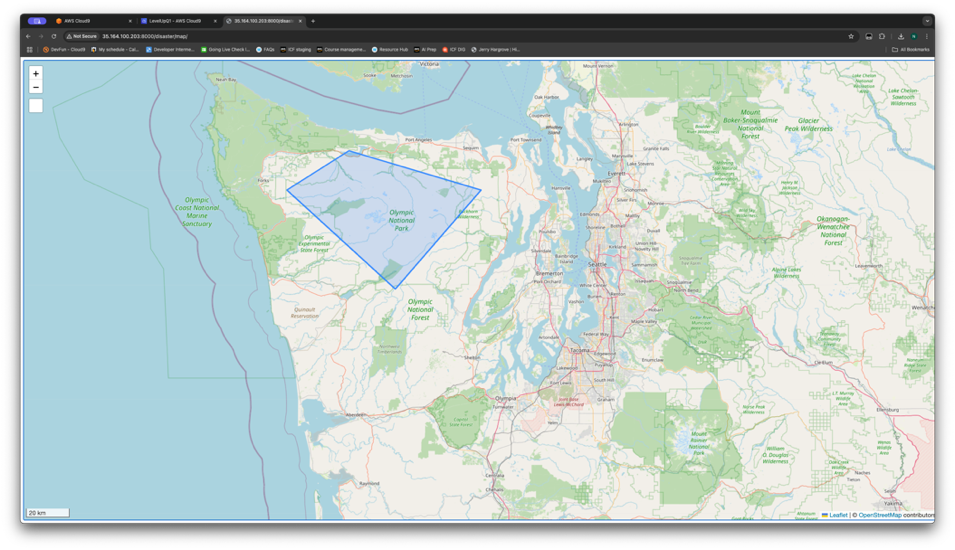


### Details Page

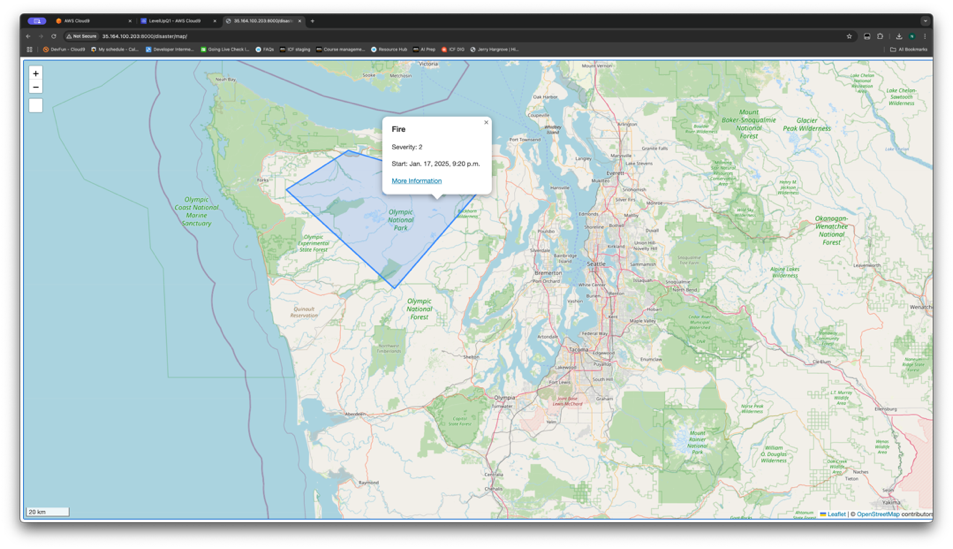


## Proof Of Concept

### Map Page



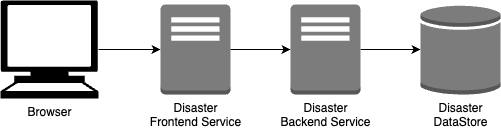
### Popup



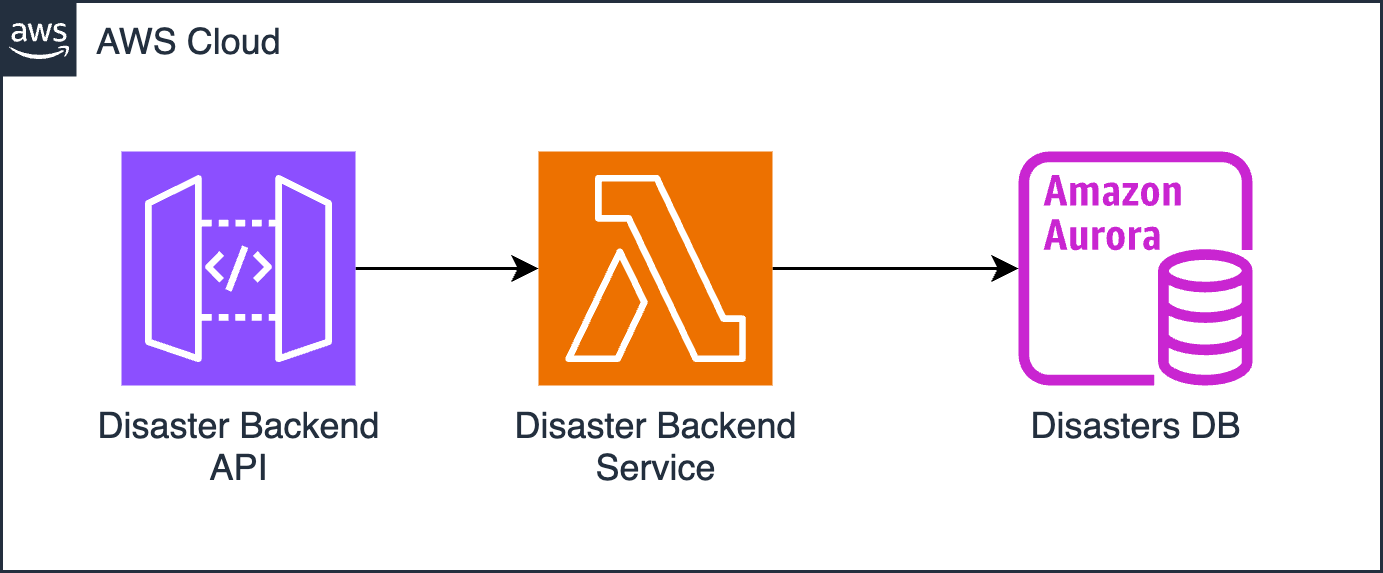
# Proposed Architecture

## High-Level Overview

Our design follows a [Three-Tier architecture](https://docs.aws.amazon.com/whitepapers/latest/serverless-multi-tier-architectures-api-gateway-lambda/welcome.html), to more loosely couple disaster information and the presentation logic.



## Architectural Overview



### Components

#### Disaster Frontend

#### Disaster Backend API

The Disaster Backend API exposes the available operations of the Disaster Backend Service as a RESTful API.

#### Disaster Backend Service

The Disaster Backend Service acts as the logic layer between the frontend and the Disasters DB. This layer handles requests for disaster data, performs queries on the disaster database to create, retrieve, update, delete, or list disaster data, and returns the data in a GeoJSON format, to be used by the frontend.

#### Disasters Database

The Disasters Database stores information related to Disasters. Disaster data includes properties such as disaster type, severity, start and end dates for each disaster. Disaster data also includes geospatial data, representing the area of impact for each disaster.

### Chosen Technologies

#### Disaster Frontend

#### Disaster Backend Service: Single AWS Lambda Function & API Gateway

* Requests are stateless
* Requests will be handled within seconds
* Fully managed scaling
* 1 million free requests / mo.
* Only pay while invoked
* 10GB memory limit is a minor concern if we have many disasters – we may need to implement pagination / tiling in the future
* Easily integrates w/ API Gateway (including proxy)
* Minimal cold starts

##### Alternative 1: Multiple AWS Lambda Functions & API Gateway

* Pros:
  + Same as single AWS Lambda Function
* Cons:
  + Code duplication
  + Segmented monitoring/logging
  + More likely to encounter cold starts

##### Alternative 2: ECS Fargate

* Pros:
  + Fully managed / serverless
* Cons:
  + More costly compared to AWS Lambda for 1st million requests / mo.

#### Backend Framework:

##### Alternative 1:

#### Disasters Database: Aurora PostgreSQL Serverless

* PostGIS extension for geospatial data
* Single-DB solution (normalization, transactional)
* Serverless – fully managed (no need to worry about increasing storage or compute capacity)

##### Alternative 1: DynamoDB

* Pros:
  + Only need simple queries
  + Schema-less: allows customization of different disaster types
* Cons:
  + Need to perform geospatial analysis manually (stretch goal)
  + Unknown number of points for polygon – can create very large attributes
  + Strong consistency costs more

##### Alternative 2: DynamoDB & Aurora

* Use DynamoDB for disaster properties, Aurora for spatial data
* Pros:
  + Benefits of both types of databases
* Cons:
  + Ensuring consistency between the two tables is complicated (especially for transactions)
  + Eventual consistency (or pay more) for properties

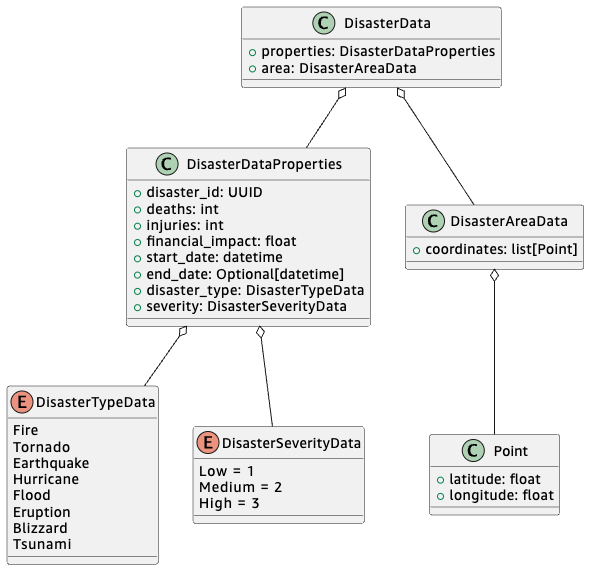
##### Alternative 3: Redis Caching

* Pros:
  + Can add geospatial queries to underlying DB
* Cons:
  + Eventual consistency
  + May need to handle large amounts of data in-memory
  + Adds additional complexity to design
  + Additional costs for caching

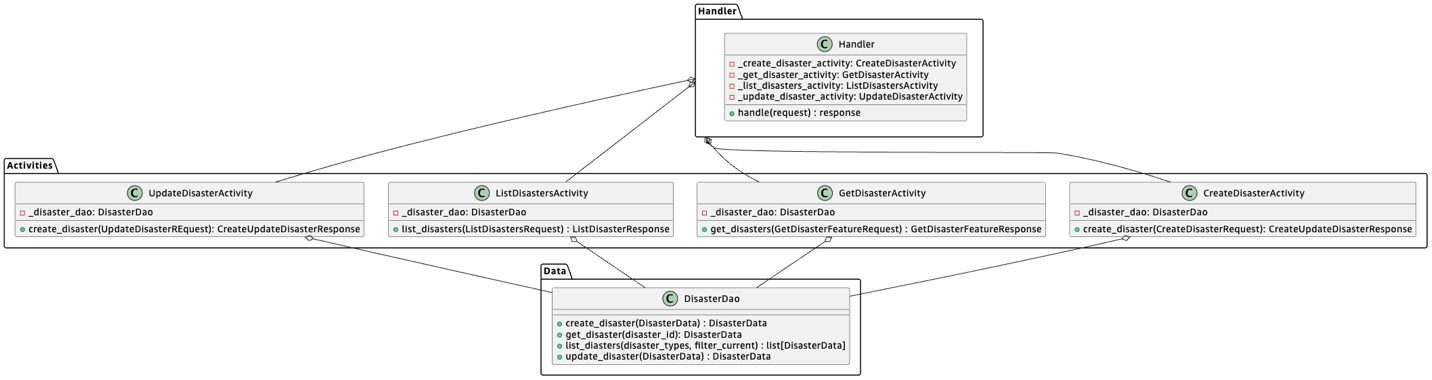
# Service Design

## Disaster Backend Service

### Internal Models



### Components

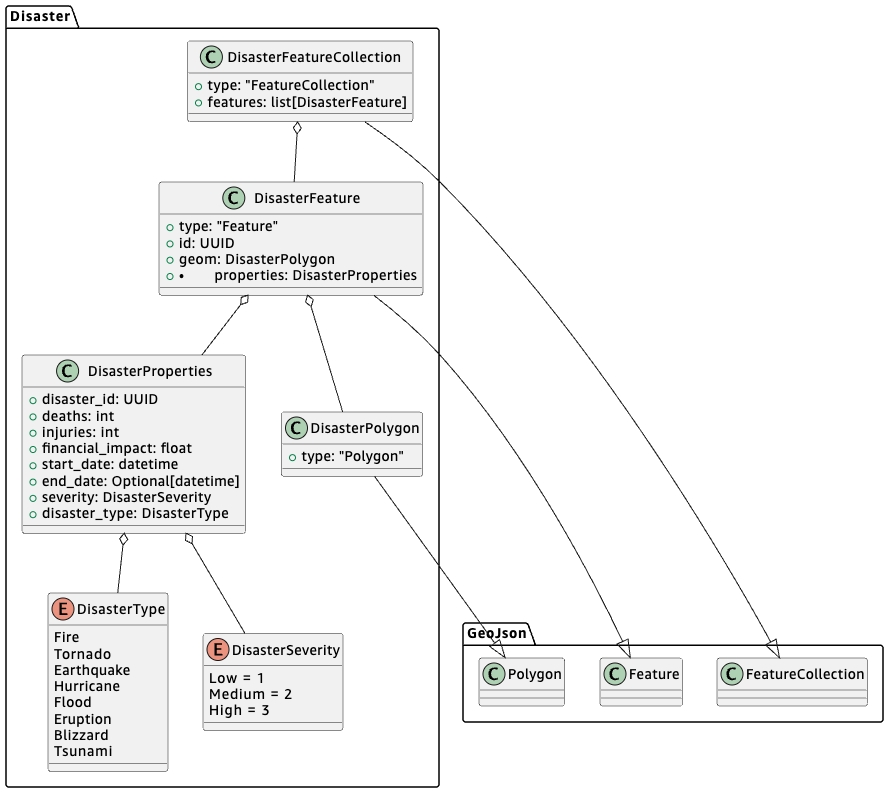


### API Design

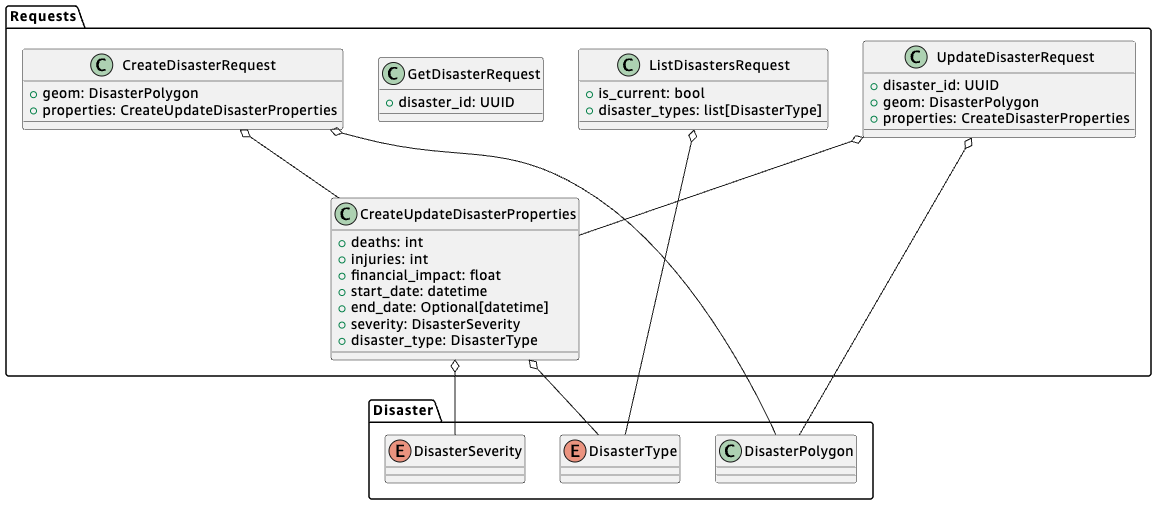
#### Public Models

##### Disaster

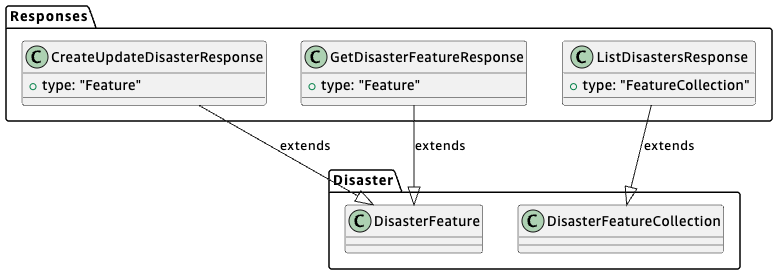
Classes in the “GeoJson” package follow the [GeoJSON specification](https://datatracker.ietf.org/doc/html/rfc7946).



##### Requests



##### Responses



#### Endpoints

##### ListDisasters

Method:

* GET

Path:

* /disaster?is\_current={is\_current}&disaster\_type={disaster\_type}&disaster\_type={disaster\_type}

Request:

* ListDisastersRequest
  + {is\_current} -> is\_current
  + {disaster\_type} -> disaster\_types
    - Multiple: Collect into a list

Response:

* Status: 200
* ListDisastersResponse
  + `features` is empty if no disasters matching query params is found

Error Responses:

* 500: internal server error

##### GetDisaster

Method:

* GET

Path:

* /disaster/{disaster\_id}

Request:

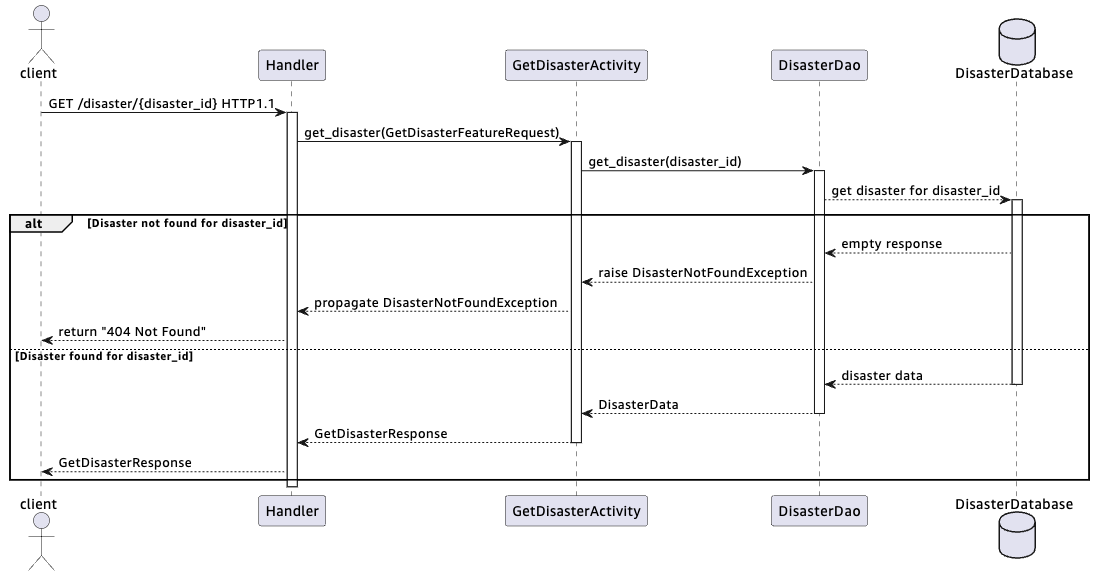
* GetDisasterRequest
  + {disaster\_id} -> disaster\_id

Response:

* Status: 200
* GetDisasterFeatureResponse

Error Responses:

* 404: no disaster found for given ID
* 500: internal server error



##### CreateDisaster

Method:

* POST

Path:

* /disaster

Header:

* Authorization: Bearer {token}

Request:

* CreateDisasterRequest (body)

Response:

* Status: 200
* CreateUpdateDisasterResponse

Error Responses:

* 401: unauthorized request
* 500: internal server error

##### UpdateDisaster

Method:

* PUT

Path:

* /disaster/{disaster\_id}

Header:

* Authorization: Bearer {token}

Path parameters:

* disaster\_id: str (UUID)

Body:

* UpdateDisasterRequest
  + {disaster\_id} -> disaster\_id
  + Body -> geom, properties

Response:

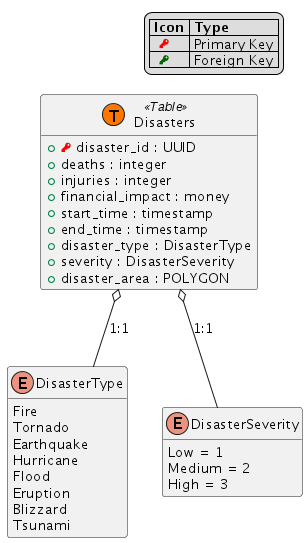
* Status: 200
* CreateUpdateDisasterResponse

Error Responses:

* 401: unauthorized request
* 404: no disaster found for given ID
* 500: internal server error

# Data Design

## Table 1: Disasters



# Development Operations

*Define what tools you will use to build, test, deploy, release, and monitor your service.*

## Code Review Process

## Deployment Process

Do you have a continuous deployment pipeline? Do you require a manual approval step? Do you plan to have a blue-green deployment strategy? What is your integration testing plan?

## Logging, Monitoring, & Metrics

What platform do you plan to use for monitoring? What alarms do you want to set, and how do you want to respond to them? Where will you store your logs? Do you want to perform any log processing?

# Tasks and Milestones

*Break down the effort this project will require into tasks and subtasks. Then group tasks into milestones.*

*Tasks should cover a vertical slice of your effort – for example, each API endpoint may constitute a task. Creating the database table may be a subtask, as would creating the API, creating the business logic, and so forth.*

*You may also choose to split some tasks or milestones by horizontal slices – maybe you want to create a separate milestone for creating the initial architecture, or one for implementing your UX.*

*If you need to perform any additional investigation, create tasks for those as well.*

## Milestone 1: <Title>

### Task 1.1: <Title>

#### Subtask 1.1.1: <Title>

# Appendix 1: Definition of Terms

|  |  |
| --- | --- |
| Term | Definition |
| Natural Disaster | Earth quakes, Tornadoes, Hurricanes, Fires, Floods, Volcano Eruptions, Blizzards, Tsunami |
| NWS | Natural Weather Service |
| SLA | “Service Level Agreement” – agreement on what will be delivered, and under what timelines. |
| URI | “Uniform Resource Identifier – a system for identifying resources. In our case, this means the path to the resource. |
| URL | “Uniform Resource Location” – a system for identifying locations of resources. Often includes a protocol, a domain name, and a path to the resource. |
| USD | United States Dollars |
| USGS | United States Geological Survey |
| UX | “User Experience” |

# Appendix 2: Citations

1. H. Butler, M. Daly, A. Doyle, S. Gillies, S. Hagen, and T. Schaub. 2016. RFC 7946: The GeoJSON Format. RFC Editor, USA, datatracker.ietf.org/doc/html/rfc7946
2. Baird, A., Bost, B., Buliani, S., Nagrani, V., Nair, A., Popat, R., & Singh, B. (2021, October 20). *AWS serverless Multi-Tier Architectures with amazon API gateway and AWS Lambda - AWS Serverless Multi-Tier Architectures with amazon API gateway and Aws Lambda*. AWS Serverless Multi-Tier Architectures with Amazon API Gateway and AWS Lambda. https://docs.aws.amazon.com/whitepapers/latest/serverless-multi-tier-architectures-api-gateway-lambda/welcome.html

# Appendix 3: Diagram Sources

## Service Design

### Disaster Backend Service

#### Internal Models

1. @startuml

2.

3. class DisasterData {

4. + properties: DisasterDataProperties

5. + area: DisasterAreaData

6. }

7.

8. class DisasterDataProperties {

9. + disaster\_id: UUID

10. + deaths: int

11. + injuries: int

12. + financial\_impact: float

13. + start\_date: datetime

14. + end\_date: Optional[datetime]

15. + disaster\_type: DisasterTypeData

16. + severity: DisasterSeverityData

17. }

18.

19. class DisasterAreaData {

20. + coordinates: list[Point]

21. }

22.

23. class Point {

24. + latitude: float

25. + longitude: float

26. }

27.

28. enum DisasterTypeData {

29. Fire

30. Tornado

31. Earthquake

32. Hurricane

33. Flood

34. Eruption

35. Blizzard

36. Tsunami

37. }

38.

39. enum DisasterSeverityData {

40. Low = 1

41. Medium = 2

42. High = 3

43. }

44.

45. DisasterData o-- DisasterDataProperties

46. DisasterDataProperties o-- DisasterTypeData

47. DisasterDataProperties o-- DisasterSeverityData

48. DisasterData o-- DisasterAreaData

49. DisasterAreaData o-- Point

50. @enduml

#### Components

1. @startuml

2.

3. package Handler {

4. class Handler {

5. - \_create\_disaster\_activity: CreateDisasterActivity

6. - \_get\_disaster\_activity: GetDisasterActivity

7. - \_list\_disasters\_activity: ListDisastersActivity

8. - \_update\_disaster\_activity: UpdateDisasterActivity

9. + handle(request) : response

10. }

11. }

12.

13. package Activities {

14.

15. class CreateDisasterActivity {

16. - \_disaster\_dao: DisasterDao

17. + create\_disaster(CreateDisasterRequest): CreateUpdateDisasterResponse

18. }

19.

20. class GetDisasterActivity {

21. - \_disaster\_dao: DisasterDao

22. + get\_disaster(GetDisasterFeatureRequest) : GetDisasterFeatureResponse

23. }

24.

25. class ListDisastersActivity {

26. - \_disaster\_dao: DisasterDao

27. + list\_disasters(ListDisastersRequest) : ListDisasterResponse

28. }

29.

30. class UpdateDisasterActivity {

31. - \_disaster\_dao: DisasterDao

32. + create\_disaster(UpdateDisasterREquest): CreateUpdateDisasterResponse

33. }

34. }

35.

36. package Data {

37.

38. class DisasterDao {

39. + create\_disaster(DisasterData) : DisasterData

40. + get\_disaster(disaster\_id): DisasterData

41. + list\_diasters(disaster\_types, filter\_current) : list[DisasterData]

42. + update\_disaster(DisasterData) : DisasterData

43. }

44. }

45.

46. Handler o-- CreateDisasterActivity

47. Handler o-- ListDisastersActivity

48. Handler o-- GetDisasterActivity

49. Handler o-- UpdateDisasterActivity

50. CreateDisasterActivity o-- DisasterDao

51. GetDisasterActivity o-- DisasterDao

52. ListDisastersActivity o-- DisasterDao

53. UpdateDisasterActivity o-- DisasterDao

54.

55. @enduml

56.

#### API Design

##### Public Models

###### Disaster Models

1. @startuml

2.

3.

4. package GeoJson {

5. class Polygon

6. class Feature

7. class FeatureCollection

8. }

9.

10. package Disaster{

11.

12. enum DisasterType {

13. Fire

14. Tornado

15. Earthquake

16. Hurricane

17. Flood

18. Eruption

19. Blizzard

20. Tsunami

21. }

22.

23. enum DisasterSeverity {

24. Low = 1

25. Medium = 2

26. High = 3

27. }

28.

29. class DisasterProperties {

30. + disaster\_id: UUID

31. + deaths: int

32. + injuries: int

33. + financial\_impact: float

34. + start\_date: datetime

35. + end\_date: Optional[datetime]

36. + severity: DisasterSeverity

37. + disaster\_type: DisasterType

38. }

39.

40.

41.

42. DisasterProperties o-- DisasterType

43. DisasterProperties o-- DisasterSeverity

44.

45.

46.

47. class DisasterPolygon {

48. + type: "Polygon"

49. }

50.

51. DisasterPolygon --|> Polygon

52.

53. class DisasterFeature {

54. + type: "Feature"

55. + id: UUID

56. + geom: DisasterPolygon

57. + properties: DisasterProperties

58. }

59.

60. DisasterFeature --|> Feature

61. DisasterFeature o-- DisasterPolygon

62. DisasterFeature o-- DisasterProperties

63.

64. class DisasterFeatureCollection {

65. + type: "FeatureCollection"

66. + features: list[DisasterFeature]

67. }

68.

69. DisasterFeatureCollection --|> FeatureCollection

70. DisasterFeatureCollection o-- DisasterFeature

71.

72. }

73.

74.

75.

76. @enduml

77.

###### Request Models

1. @startuml

2.

3. package Disaster {

4. class DisasterPolygon

5. enum DisasterType

6. enum DisasterSeverity

7. }

8.

9.

10. package Requests {

11.

12. class CreateUpdateDisasterProperties {

13. + deaths: int

14. + injuries: int

15. + financial\_impact: float

16. + start\_date: datetime

17. + end\_date: Optional[datetime]

18. + severity: DisasterSeverity

19. + disaster\_type: DisasterType

20. }

21.

22. CreateUpdateDisasterProperties o-- DisasterType

23. CreateUpdateDisasterProperties o-- DisasterSeverity

24.

25. class CreateDisasterRequest {

26. + geom: DisasterPolygon

27. + properties: CreateUpdateDisasterProperties

28. }

29.

30. CreateDisasterRequest o-- DisasterPolygon

31. CreateDisasterRequest o-- CreateUpdateDisasterProperties

32.

33. class UpdateDisasterRequest {

34. + disaster\_id: UUID

35. + geom: DisasterPolygon

36. + properties: CreateDisasterProperties

37. }

38.

39. UpdateDisasterRequest o-- DisasterPolygon

40. UpdateDisasterRequest o-- CreateUpdateDisasterProperties

41.

42. class GetDisasterRequest {

43. + disaster\_id: UUID

44. }

45.

46.

47. class ListDisastersRequest {

48. + is\_current: bool

49. + disaster\_types: list[DisasterType]

50. }

51.

52. ListDisastersRequest o-- DisasterType

53.

54. }

55.

56. @enduml

###### Response Models

1. @startuml

2.

3. package Disaster {

4. class DisasterFeature

5. class DisasterFeatureCollection

6. }

7.

8. package Responses {

9.

10. class CreateUpdateDisasterResponse {

11. + type: "Feature"

12. }

13.

14. CreateUpdateDisasterResponse --|> DisasterFeature: extends

15.

16. class GetDisasterFeatureResponse {

17. + type: "Feature"

18. }

19.

20. GetDisasterFeatureResponse --|> DisasterFeature: extends

21.

22. class ListDisastersResponse {

23. + type: "FeatureCollection"

24. }

25.

26. ListDisastersResponse --|> DisasterFeatureCollection: extends

27.

28. }

29. @enduml

30.

#### Sequence Diagrams

##### GetDisaster

1. @startuml

2.

3. actor client

4. participant Handler

5. participant GetDisasterActivity

6. participant DisasterDao

7. database DisasterDatabase

8.

9. client -> Handler: GET /disaster/{disaster\_id} HTTP1.1

10. activate Handler

11. Handler -> GetDisasterActivity :get\_disaster(GetDisasterFeatureRequest)

12. activate GetDisasterActivity

13. GetDisasterActivity -> DisasterDao: get\_disaster(disaster\_id)

14. activate DisasterDao

15. DisasterDao --> DisasterDatabase: get disaster for disaster\_id

16. activate DisasterDatabase

17. alt Disaster not found for disaster\_id

18. DisasterDao <-- DisasterDatabase: empty response

19. GetDisasterActivity <-- DisasterDao: raise DisasterNotFoundException

20. Handler <-- GetDisasterActivity: propagate DisasterNotFoundException

21. client <-- Handler: return "404 Not Found"

22. else Disaster found for disaster\_id

23. DisasterDao <-- DisasterDatabase: disaster data

24. deactivate DisasterDatabase

25. GetDisasterActivity <-- DisasterDao: DisasterData

26. deactivate DisasterDao

27. Handler <-- GetDisasterActivity: GetDisasterResponse

28. deactivate GetDisasterActivity

29. client <-- Handler: GetDisasterResponse

30. end

31. deactivate Handler

32.

33. @enduml

## Data Design

### Table 1: Disasters

1. @startuml

2. ! $PrimaryKeyIcon = "<color:red><&key></color>"

3. ! $ForeignKeyIcon = "<color:darkgreen><&key></color>"

4. ! $DefaultType = ""

5. hide methods

6. !unquoted procedure Table($table)

7. entity $table << (T,#FF7700) Table >>

8. !endprocedure

9. !unquoted procedure PrimaryKey($val, $type=$DefaultType)

10. Type($val, $type, $PrimaryKeyIcon)

11. !endprocedure

12. !unquoted procedure ForeignKey($val, $type=$DefaultType)

13. Type($val, $type, $ForeignKeyIcon)

14. !endprocedure

15. !unquoted procedure Attribute($val, $type=$DefaultType)

16. Type($val, $type)

17. !endprocedure

18. !unquoted procedure Type($val, $type=$DefaultType, $icon="")

19. !if ($type=="")

20. +$icon $val

21. !else

22. +$icon $val : $type

23. !endif

24. !endprocedure

25.

26. Table(Disasters) {

27. PrimaryKey(disaster\_id, UUID)

28. Attribute(deaths, integer)

29. Attribute(injuries, integer)

30. Attribute(financial\_impact, money)

31. Attribute(start\_time, timestamp)

32. Attribute(end\_time, timestamp)

33. Attribute(disaster\_type, DisasterType)

34. Attribute(severity, DisasterSeverity)

35. Attribute(disaster\_area, POLYGON)

36.

37. }

38.

39. enum DisasterType {

40. Fire

41. Tornado

42. Earthquake

43. Hurricane

44. Flood

45. Eruption

46. Blizzard

47. Tsunami

48. }

49.

50. enum DisasterSeverity {

51. Low = 1

52. Medium = 2

53. High = 3

54. }

55.

56. Disasters o-- DisasterType : 1:1

57. Disasters o-- DisasterSeverity : 1:1

58.

59.

60. legend top right

61. |= Icon |= Type |

62. | $PrimaryKeyIcon | Primary Key |

63. | $ForeignKeyIcon | Foreign Key |

64. end legend

65. @enduml

66.