# United States Natural Disaster Service

Design Document

Table of Contents

[United States Natural Disaster Service 1](#_Toc188609012)

[Problem Statement 3](#_Toc188609013)

[Use Cases 3](#_Toc188609014)

[U1: Disaster Details 3](#_Toc188609015)

[U2: Topological Map 3](#_Toc188609016)

[U3: Create Disaster 3](#_Toc188609017)

[U4: Update Disaster 3](#_Toc188609018)

[Project Scope 4](#_Toc188609019)

[In Scope 4](#_Toc188609020)

[Out of Scope 4](#_Toc188609021)

[Open Questions 5](#_Toc188609022)

[Functional Requirements 5](#_Toc188609023)

[U1: Disaster Details 5](#_Toc188609024)

[FR 1.1: valid\_inputs\_get\_disaster\_returns\_disaster\_details 5](#_Toc188609025)

[FR 1.2: no\_matching\_disaster\_get\_disaster\_not\_found\_error 5](#_Toc188609026)

[U2: Topological Map 6](#_Toc188609027)

[FR 2.1: current\_disasters\_map\_current\_disasters\_returns\_disasters 6](#_Toc188609028)

[FR 2.2: no\_current\_disasters\_map\_current\_disasters\_returns\_empty\_map 6](#_Toc188609029)

[FR 2.3: click\_on\_disaster\_area\_renders\_popup 6](#_Toc188609030)

[U3: Create Disaster 7](#_Toc188609031)

[FR 3.1: valid\_administrator\_create\_disaster\_creates\_disaster 7](#_Toc188609032)

[FR 3.2: unauthorized\_user\_create\_disaster\_forbidden\_error 7](#_Toc188609033)

[FR 3.3: missing\_data\_create\_disaster\_bad\_request\_error 8](#_Toc188609034)

[FR 3.4: future\_start\_date\_create\_disaster\_bad\_request\_error 8](#_Toc188609035)

[FR 3.5: invalid\_disaster\_type\_create\_disaster\_bad\_request\_error 8](#_Toc188609036)

[FR 3.6: invalid\_severity\_create\_disaster\_bad\_request\_error 8](#_Toc188609037)

[FR 3.7 negative\_deaths\_create\_disaster\_bad\_request\_error 9](#_Toc188609038)

[FR 3.8: negative\_injuries\_create\_disaster\_bad\_request\_error 9](#_Toc188609039)

[FR 3.9: negative\_financial\_impact\_create\_disaster\_bad\_request\_error 9](#_Toc188609040)

[FR 3.10: too\_many\_decimals\_financial\_impact\_create\_disaster\_bad\_request\_error 9](#_Toc188609041)

[FR 3.11: invalid\_location\_create\_disaster\_bad\_request\_error 9](#_Toc188609042)

[FR 3.12: invalid\_area\_create\_disaster\_bad\_request\_error 10](#_Toc188609043)

[U4: Update Disaster 10](#_Toc188609044)

[FR 4.1: valid\_administrator\_update\_disaster\_creates\_disaster 10](#_Toc188609045)

[FR 4.2: unauthorized\_user\_update\_disaster\_forbidden\_error 10](#_Toc188609046)

[FR 4.3: disaster\_does\_not\_exist\_update\_disaster\_not\_found\_error 11](#_Toc188609047)

[FR 4.4: missing\_data\_update\_disaster\_bad\_request\_error 11](#_Toc188609048)

[FR 4.5: future\_start\_date\_update\_disaster\_bad\_request\_error 11](#_Toc188609049)

[FR 4.6: future\_end\_date\_update\_disaster\_bad\_request\_error 12](#_Toc188609050)

[FR 4.7: invalid\_disaster\_type\_update\_disaster\_bad\_request\_error 12](#_Toc188609051)

[FR 4.8: invalid\_severity\_update\_disaster\_bad\_request\_error 12](#_Toc188609052)

[FR 4.9: negative\_deaths\_update\_disaster\_bad\_request\_error 12](#_Toc188609053)

[FR 4.10: negative\_injuries\_update\_disaster\_bad\_request\_error 13](#_Toc188609054)

[FR 4.11: negative\_financial\_impact\_update\_disaster\_bad\_request\_error 13](#_Toc188609055)

[FR 4.12: too\_many\_decimals\_financial\_impact\_update\_disaster\_bad\_request\_error 13](#_Toc188609056)

[FR 4.13: invalid\_location\_update\_disaster\_bad\_request\_error 13](#_Toc188609057)

[FR 4.14: invalid\_area\_update\_disaster\_bad\_request\_error 13](#_Toc188609058)

[UX Design 14](#_Toc188609059)

[Mockups 14](#_Toc188609060)

[Map Page 14](#_Toc188609061)

[Popup 14](#_Toc188609062)

[Details Page 15](#_Toc188609063)

[Proof Of Concept 15](#_Toc188609064)

[Map Page 15](#_Toc188609065)

[Popup 15](#_Toc188609066)

[Proposed Architecture 16](#_Toc188609067)

[Architectural Overview 16](#_Toc188609068)

[Architectural Diagram 16](#_Toc188609069)

[Chosen Technologies 16](#_Toc188609070)

[Technology 1: <Name> 16](#_Toc188609071)

[Alternative Technologies 16](#_Toc188609072)

[Alternative 1: <Name> 16](#_Toc188609073)

[Service Overview 16](#_Toc188609074)

[API Design 16](#_Toc188609075)

[Public Models 16](#_Toc188609076)

[Model 1: <Name> 16](#_Toc188609077)

[Endpoints 17](#_Toc188609078)

[Endpoint 1: <URI> : <HTTP Method> 17](#_Toc188609079)

[Data Design 17](#_Toc188609080)

[Table 1: <Name> 17](#_Toc188609081)

[Development Operations 17](#_Toc188609082)

[Source Control & Standards 18](#_Toc188609083)

[Coding Language(s) & Standards 18](#_Toc188609084)

[<Component> : <Language 1, ex: Python 3.11> 18](#_Toc188609085)

[Containerization / Build Platform 18](#_Toc188609086)

[Component 19](#_Toc188609087)

[Build Platform 19](#_Toc188609088)

[Component 1 19](#_Toc188609089)

[Ex: Docker 19](#_Toc188609090)

[Testing Details 19](#_Toc188609091)

[Code Review Process 19](#_Toc188609092)

[Deployment Process 19](#_Toc188609093)

[Logging, Monitoring, & Metrics 19](#_Toc188609094)

[Tasks and Milestones 19](#_Toc188609095)

[Milestone 1: <Title> 20](#_Toc188609096)

[Task 1.1: <Title> 20](#_Toc188609097)

[Appendix 1: Definition of Terms 20](#_Toc188609098)

[Appendix 2: References & Citations 20](#_Toc188609099)

[Appendix 3: Design Document Resources <You can delete later> 21](#_Toc188609100)

# 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
  + Location
    - Center (Lattitude & Longitude)
  + 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 an 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

*Use this space for any questions you still need to answer as you update/review this document.*

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?

# Functional Requirements

## 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
  + Location
    - Center (Lattitude & Longitude)
  + Area
    - Polygon (Points – Lattitude & Longitude)
  + 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

### 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

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

GIVEN:

* An authenticated, authorized administrator
* A valid disaster, including:
  + Location
    - Center Point (Lattitude & Longitude)
  + 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
  + Location
    - Center Point (Lattitude & Longitude)
  + 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:
  + Location
    - Center Point (Lattitude & Longitude)
  + 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

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

GIVEN:

* An authenticated, authorized administrator
* A valid disaster, including:
  + ID
  + Location
    - Center Point (Lattitude & Longitude)
  + 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
  + Location
    - Center Point (Lattitude & Longitude)
  + 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

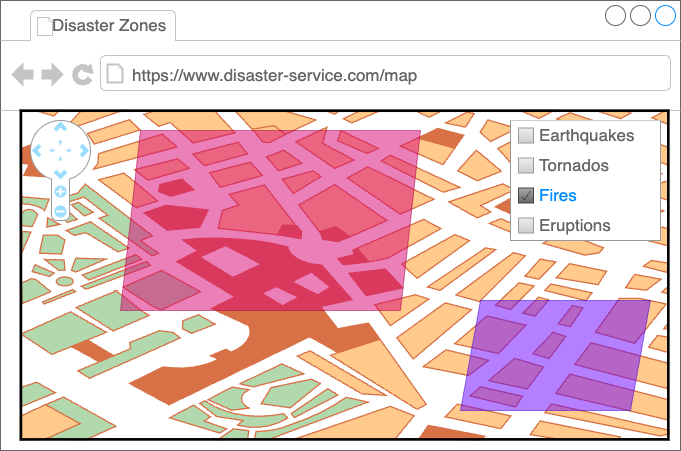
THEN:

* Return a 400 Bad Request error

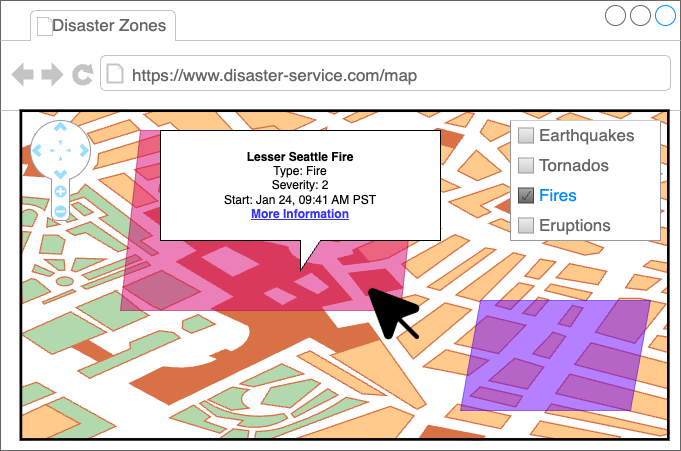
# UX Design

## Mockups

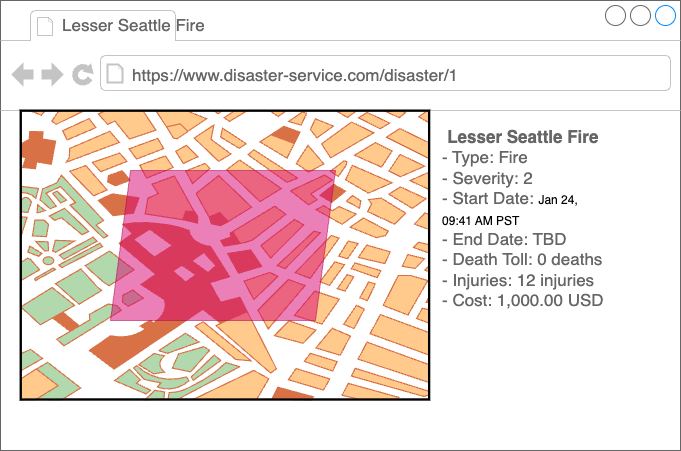
### Map Page



### Popup

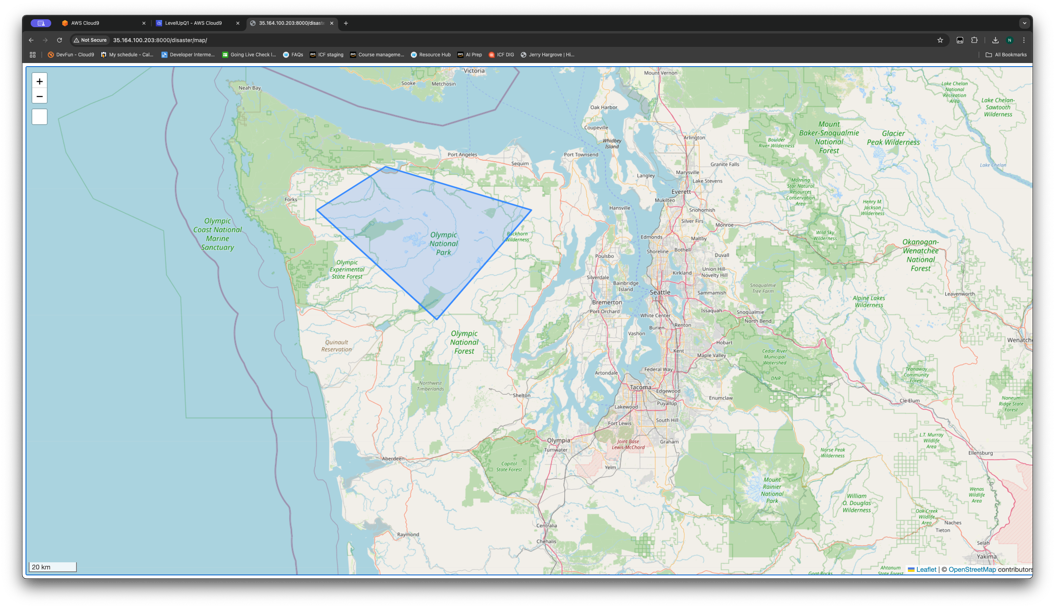


### Details Page

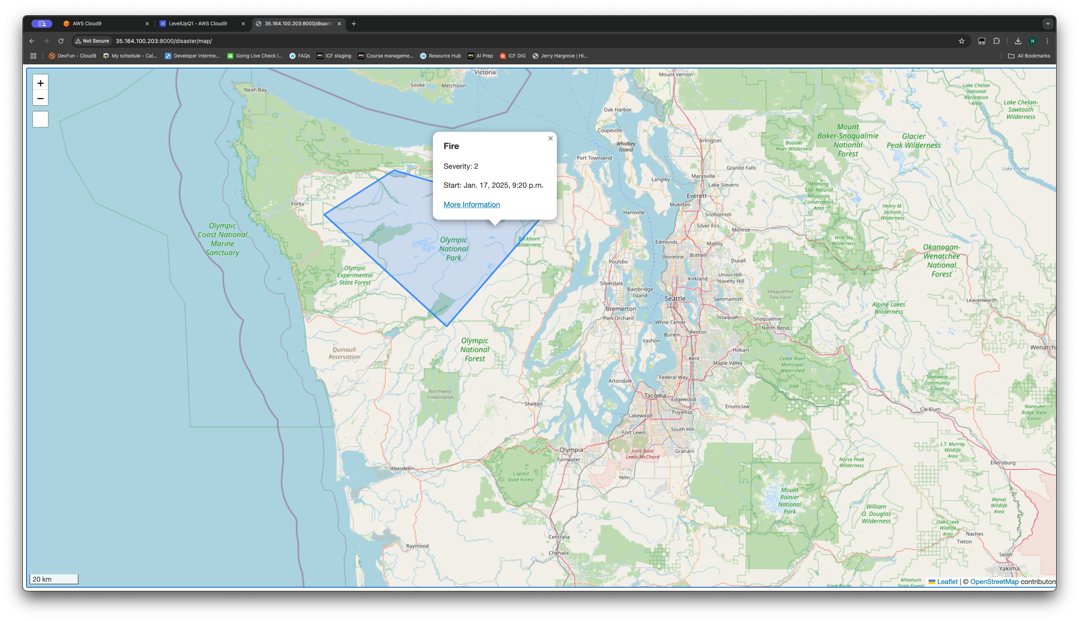


## Proof Of Concept

### Map Page



### Popup



# Proposed Architecture

*Describe broadly how you are proposing to solve for the requirements you described earlier.*

## Architectural Overview

*This should include what web services or other technologies you plan to use, as well as an architecture diagram.*

*You should describe your technical decisions, and argue why the technologies you chose are reasonable. Describe how each component will behave as part of your system.*

## Architectural Diagram

<Diagram>

## Chosen Technologies

### Technology 1: <Name>

## Alternative Technologies

*Describe alternative technologies or strategies, and why you did not choose to use them.*

### Alternative 1: <Name>

## Service Overview

*This may include class diagram(s) showing what components you are planning to build.*  
*You should argue why this architecture (organization of components) is reasonable. That is, why it represents a good data flow and a good separation of concerns. Where applicable, argue why this architecture satisfies the stated requirements.*

# API Design

## Public Models

*Define the data models your service will expose in its responses.*

### Model 1: <Name>

<Class Diagram>

## Endpoints

### Endpoint 1: <URI> : <HTTP Method>

*Describe the behavior of the first endpoint you will build into your service API. This should include what data it requires, what data it returns, and how it will handle any known failure cases. You should also include a sequence diagram showing how a user interaction goes from user to website to service to database, and back.*  
  
*(You should have a separate section for each of the endpoints you are expecting to build)*

Request: <Data Model>

Response: <Data Model>

Error Responses:

* <Error Code>: <Condition>

Sequence Diagram:

<Diagram> (suggest also including source code for it, if possible)

# Data Design

*Define the tables you will need for the data your service will use. It may be helpful to first think of what objects your service will need, then translate that to a table structure. This should include any entity-relationship diagrams to describe how your different tables will be related.*

<Entity-Relationship Diagram(s)>

## Table 1: <Name>

Key Schema:

* <Attribute name> : <Type>

Attributes

* <Attribute name> : <Type>

Foreign Keys:

* <Foreign Table> : <Attribute name> : <Type>

# Development Operations

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

*Some examples of what to include:*

## Source Control & Standards

*Example:*

*We will use Git for our source control platform.*

*We will create 1 feature branch for each milestone, and 1 branch from the milestone branch for each task. Changes from tasks will be pulled to the milestone branch as they are completed. Changes in milestone branches will be pulled to `mainline` on a regular cadence (1 business week, on Mondays at 1pm PST).*

*To maintain clean branches, we will use `rebase` instead of `merge` for pulling changes to other branches.*

*We will follow the following commit message template:[[1]](#footnote-1)*

1. # Title: Summary, imperative, start upper case, don't end with a period

2. # No more than 50 chars. #### 50 chars is here: #

3.

4. # Remember blank line between title and body.

5.

6. # Body: Explain \*what\* and \*why\* (not \*how\*). Include task ID (Jira issue).

7. # Wrap at 72 chars. ################################## which is here: #

8.

9.

## Coding Language(s) & Standards

### <Component> : <Language 1, ex: Python 3.11>

* We will follow the [PEP 8 style guidelines](https://peps.python.org/pep-0008/)
* We will follow the [PEP 257 docstring conventions](https://peps.python.org/pep-0257/)
* We will enforce conventions using the following tools:
  + [Flake8](https://flake8.pycqa.org/en/latest/) for linting
  + [isort](https://pycqa.github.io/isort/) for import sorting
  + [black](https://black.readthedocs.io/en/stable/) for formatting

## Containerization / Build Platform

|  |  |
| --- | --- |
| Component | Build Platform |
| Component 1 | Ex: Docker |
|  |  |

## Testing Details

*Ex:*

*We will use the `unittest` library for unit testing our python components.*

*We will use `pytest` for integration testing our python components.*

*We will maintain 95% code coverage for all python components.*

## Code Review Process

*How many required approvals? SLA time on CR feedback. How long are approved CRs allow to sit before being pushed to the respective branches?*

## 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?

## 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

*Use this space to define terms or acronyms that your audience may not be familiar with.*

|  |  |
| --- | --- |
| Term | Definition |
| CR | Code Review |
| linting | The process of using a lint tool, or a static code analyzer, to automatically check source code for errors. |
| Natural Disaster | Earth quakes, Tornadoes, Hurricanes, Fires, Floods, Volcano Eruptions, Blizzards, Tsunami |
| NWS | Natural Weather Service |
| PEP | “Python Enhancement Proposals” |
| 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: References & Citations

1. Cbeams. “How to Write a Git Commit Message.” *Cbeams*, cbeams, 27 May 2023, cbea.ms/git-commit/.
2. “Use a Git Commit Message Template to Write Better Commit Messages.” *Gist*, lisawolderiksen, gist.github.com/lisawolderiksen/a7b99d94c92c6671181611be1641c733.
3. van Rossum , Guido, et al. “PEP 8 – Style Guide for Python Code.” *Python EnhancementProposals (PEPs)*, 5 July 2001, peps.python.org/pep-0008/.
4. Goodger, David, and Guido van Rossum. “PEP 257 – Docstring Conventions.” *Python Enhancement Proposals (PEPs)*, 29 May 2001, peps.python.org/pep-0257/.
5. “Flake 8: Your Tool for Style Guide Enforcement.” *Flake8*, flake8.pycqa.org/en/latest/#.
6. “Isort.” *Isort*, pycqa.github.io/isort/.
7. “The Uncompromising Code Formatter.” *Black 24.8.0 Documentation*, black.readthedocs.io/en/stable/.

# Appendix 3: Design Document Resources <You can delete later>

*Here are some resources to help you create your design document.*

|  |  |
| --- | --- |
| Site | Use(s) |
| plantuml.com | Sequence/Class/Entity Relationship Diagrams |
| https://app.diagrams.net/ | Architectural Diagrams / Mockups / Other Diagrams |
| https://docs.aws.amazon.com/ | AWS Documentation |
| https://aws.amazon.com/what-is/restful-api/ | What is a RESTful API? |

1. Based on “Use a Git Commit Message Template to Write Better Commit Messages.” Gist, lisawolderiksen, gist.github.com/lisawolderiksen/a7b99d94c92c6671181611be1641c733. [↑](#footnote-ref-1)