Disaster Tap

Allen Zohrabians, Emmanoel Dermkrdichyan, Samuel Agazaryan, Liana Saryan, Akul Kumar*
{allen.zohrabians.281,emmanoel.dermkrdichyan.584,samuel.agazaryan.332,liana.saryan.214,akul.kumar.268}@my.csun.edu
Department of Computer Science
California State University, Northridge
Northridge, California

ABSTRACT

Real-time disaster information plays an important role in increasing the preparation and awareness of communities in order to reduce disaster damages in terms of the loss of life and/or property. California would benefit from these systems due to how prone the state is to such occurrences. However, for existing real-time disaster applications, issues arise in managing and obtaining the real-time data early on after a natural disaster occurs. More specifically, as this data can be quite large and comes in very frequently. As our major contribution, we attempted to tackle the issue of proper and efficient management for real-time data in the hopes of having a continuous flow of data to users unimpeded by the issues that plague real-time data handling. In this application, HTML, CSS, JavaScript, the NASA EONET API, and the Google Maps API were used to build a storage optimized real-time disaster map that provides individuals in California with the response information relevant to ensuring their safety.

KEYWORDS

Real-Time Data Visualization, Natural Disaster, Geographical Information, NASA EONET, Google Maps

1 INTRODUCTION

In the past 50 years, the state of California has declared a total of 309 State of Emergency Declarations, 30% of which have been accounted for by fires, and 7% by earthquakes [6]. Within this period of time, natural disasters have been the cause of 798 human deaths and 25,424 injuries in the state, inflicting a total cost of 56.2 billion dollars worth of damages. A real-time disaster system which provides relevant response information is imperative to mitigate these destruction costs and ensure the safety of individuals in the occurrence of a natural disaster. Existing problems faced by current implementations of such systems include shortcomings within the organization of their storage information and the lack of ability to communicate disaster information in rapid response times [10]. In order to limit damage expenses and reduce the amount of casualties and injuries imposed by natural disasters, we built a storage optimized, real-time disaster software map, that would provide individuals in California with the proper response information required to ensure their safety. NASA's EONET API was used for the collection of disaster information, the Google Maps API for visualization and construction of the map itself, and the website for user interaction and displaying the application. Upon the occurrence of a wildfire, earthquake, or other natural disaster, the data collection API provides all relevant information in regards to that event, which, in turn, will be pulled by the Google Maps API and displayed to the user with the use of plugins and visual aids. Ultimately, users will have the ability to request the disaster information and respond accordingly to reach safety.

The rest of this paper is structured as follows: Section 2 presents our collection of related works that have helped prepare us for taking on this assignment whether that be in terms of understanding natural disaster information or map and application literature. Section 3 explores the design aspect which covers our plans for the implementation of the application itself and the APIs for data collection as well as our map. It also provides an overview of our methodologies involving use cases and design strategies. Section 4 goes into detail about the techniques, algorithms, and strategies utilized throughout the duration of our experiments. Section 5 reflects on the setup, design, and results of our experiments. Section 6 concludes our project in its current version and alludes to potential subsequent extensions.

2 RELATED WORKS

This section presents the related work providing clarity in the context of our project on the subjects of application development, map development, and the intersection of the two.

2.1 Map Development

Map development works pertain to the areas regarding map manipulation itself. This includes manipulating map elements, application design specifically involving maps, and work with disaster logistics for maps.

Hu et al. worked with map 'mashups' and attempted to solve problems that concerned it [4]. A map 'mashup' is one which aggregates data from different sources; The result is a new service but at the risk of possibly using non secured data and an absence of complex functions. A new mapping application idea was proposed which used the Google Maps API and SQL which would also include complex data manipulation functionalities at the user's disposal including SQL scripts. The application's use of a secured database also removed the risk vector of non secured data. For our project, we planned to also use a SQL database to store data. This work's description of database usage as well as functionalities such as SQL scripts for data manipulation was initially within the scope of our project prior to our decision of electing not to proceed with a database since the benefits of its use would not be necessary for this project in particular.

Derrough also discussed research on manipulating map elements [1]. He detailed how to create markers and vector shapes as well as

*All authors contributed equally to this research.

COMP490 Senior Design Project, California State University, Northridge,

display data from various providers with a step-by-step approach. Derrough delved into the Leaflet API and analyzed efficient techniques for map manipulation to aid developers that wanted to create eye-catching maps with mobile compatibility. The primary technology used by the author was GeoJSON, which is an open standard format designed for representing simple geographical features along with their non spatial attributes [2]. We believed this to be extremely valuable in our groups' approach to the natural disaster map application. The plans of implementing the Leaflet API within our software to communicate between the Android application and the SQL database eventually changed as we got rid of the database, and switched from Leaflet to Google Maps as well as from Android mobile to a website. However, the methods of Derrough still served as an excellent reference for our data visualization throughout the process of development.

Erskine and Gregg proposed a new crowd sourced real-time disaster map [3]. While they used a baseline map from disaster planning initiatives, those near or effected by the disaster could use a mobile application to collect data about the disaster in many different forms, including audio, text and photographic information. They argued that through the use of crowd sourcing, more information is available to the proper authorities shortly after a disaster occurs. Their use of a base map for research can be derived from geographical data prior to the disaster, and as such in our project although we do not plan to use a crowd sourced approach we will need an accurate map to display data over, which is subject to change depending on the disaster and the intensity of it.

2.2 Application Development

Application development involves work which discusses general application development for mobile devices. Application development for natural disasters as well as evacuation research and approaches for developing map applications for mobile devices are discussed.

Kolesnikov and Kikin analyzed the differences in developing map applications within different environments and platforms: web-based, native development environment, and an hybrid implementation [5]. The goal of this work was to compare performance as well as functionality between the options to see their advantages and disadvantages. Their results found that in terms of functionality and development rates, the native approach was superior. Processing data and data visualization was hindered especially on the hybrid approach as compared to the native approach; They pointed to the lack of an integrated database management system as the likely culprit for this.

Pojee et al. discussed their design of an Android application, which ensured public access to safety by providing real-time disaster alerts and dynamically routing impacted individuals to safety [8]. In order to guarantee the validity of their alerts, data was pulled from the National Disaster Management Authority of India. The authors utilized a hypothetical tsunami threat to generate an alert that informed the user of the disaster, location, affected areas and level of severity. Clicking on the alert allowed users to share their location and navigate to safety. A dynamic tracking algorithm was activated to fetch the geo-spatial coordinates of both users and nearby safe houses. They found that this system was very robust due to predetermined components such as the positioning of safe locations. It should however be noted that their navigating algorithm depended on two factors: distance from the user's current location and the maximum capacity of a safe house. This work touched on the collection of natural disaster data as well as discussed Android application development for the sake of real-time disaster analysis, similarly to our main topic of research. However with this we on the other hand do not plan on implementing routing for actual use of victims of natural disasters for escape from said disasters.

3 DESIGN

As shown in Fig. 1, this project is composed of 3 main components: the Disaster Tap website, Google Maps API, and Data Collection API. The data collection API will bring in information about the disasters organized based on location and type of disaster. Once the disasters are sorted according to the category they fall under, they will be pulled and displayed on a live map utilizing the Google Maps API. This map is then framed inside of the website to be delivered to the application's userbase.

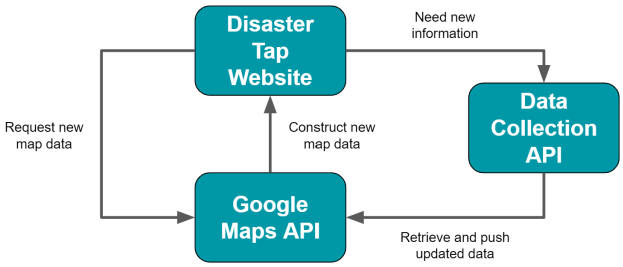

Figure 1: Functionality framework - Displays the relationship between the Disaster Tap website and the APIs.

3.1 Google Maps API

Our software utilizes the Google Maps API as our map framework, which plays an important role in the software as it also allows us to place markers and polygons on our map. Originally, we planned to use the Leaflet API for our map framework; An open-source library for interactive maps. This API enables us to visualize the data onto a map by allowing us to insert pop ups, images, and custom map projections. Leaflet is very user friendly in that it allows the user to simply move around and zoom into different areas with the swipe of a finger. This API also has many plugins to aid in visualizing data, for example a heat map plugin to show concentration in a certain area. As shown in Fig. 2, data would be pulled from our database and displayed directly onto the map with the use of the plugins. Once the data has been collected we can also send out notifications to users that are in the zone affected by a disaster by getting the users location data.

After a lot of consideration the team decided that the best option for this project would be to go with the Google Maps API instead. The main reasons being that there were less constraints for the number of calls that could be made and also more support for a web based project. With the large amounts of documentation that

Google provides with the API, it makes working on the project much easier. The Google Maps API is very similar to Leaflet in that it also allows us to insert popups, images, custom map projections, alerts and other features that allow us to visualize data.

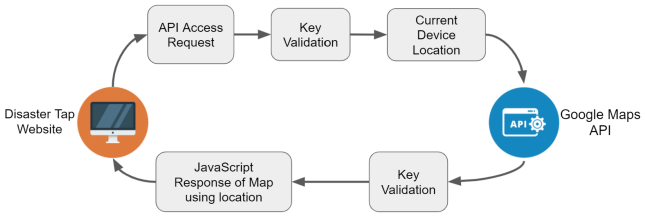

Figure 2: Google Maps API diagram: Overview of the Google Maps API design and implementation.

3.2 SQL Database

The database component initially planned for our project was responsible for processing, storing and retrieving the data that is generated for real-time natural disasters and pushing this information onto the Google Maps API.

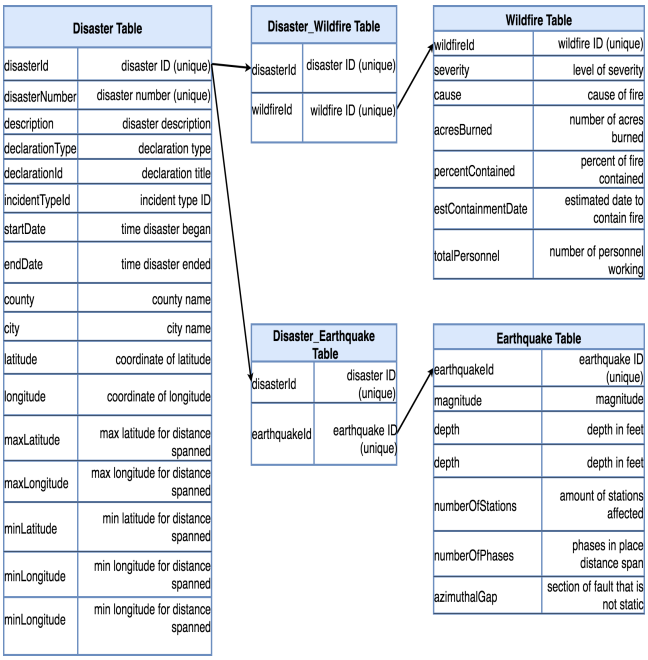

Figure 3: Database Design: Overview of the database design and implementation.

As presented by Fig. 3, the design for our database consisted of 3 main tables, in addition to 2 linking tables. General metadata for the disaster, including descriptions, declaration types, location information, as well as a time frame of their occurrences is stored in the Disaster Table. Since our application is meant to map the most common natural disasters occurring in California, wildfires and earthquakes, we decided to convey their unique attributes in separate tables. Consequently, the Wildfire Table contains information of fields specific to wildfires, such as severity, cause and acres burned, while the Earthquake Table contains the data specific to earthquakes, such as magnitude, depth and number of phases. Linking tables, Disaster Wildfire Table and Disaster Earthquake Table are used to normalize the database schema.

MySQL was the intended relational database management system to be utilized. As a platform independent database service that can be downloaded for free from the Internet, MySQL can be easily accessed and integrated as a major component. Given that the collection, organization and instant retrieval of the real-time disaster data is essential to ensuring rapid delivery of disaster information to the user, our database will be optimized to meet these requirements. Giacomo states that the combination of APIs and SQL establishes nearly complete interoperability between the database and a client [2]. By utilizing the capabilities of relational database management systems, which extend to ensuring easy access to and from APIs written in different languages, and storing large amounts of data, supporting the tremendous amount of data that is pulled from the NASA EONET API as well as relaying this information to the Google Maps API could be accomplished simply and efficiently.

3.3 Web Application

The web application component connects each piece of the software together, as was initially the role of the Android Application before transitioning. The actors which interact with the application are the Google Maps API and the users themselves. The application requests new data from the Google Maps API which in turn would obtain data from the SQL database initially. This information is sent back to the application and updates the displayed data on the map. The application can then display this data depending on the type of disaster/filter designated by the user. An interface that ensures ease of use is mandatory for a product that relies on the consistent spread of information in the countless geographical climates throughout the world. In an experiment conducted by Osaragi et al. in which local residents collect disaster-information by use of their web application system under the assumption that a large earthquake has occurred, the results confirm that the elapsed time required for synchronization of posting disaster-information among users increases to less than 50 milliseconds even if the number of users increases to a certain degree [7]. In other words, as long as there is a cloud server utilized in equivalence to the one provided for this experiment, it becomes possible to support the activities of emergency vehicles without the loss of real time property.

The team decided to proceed with the project through a website due to multiple reasons. Firstly, an increase in simplicity for integration of the different parts of the project. Also, unlike some other APIs, EONET does not require an API key and as such requests are not tracked and called locally by the user's browser. This means that our original concern of making too many calls was no longer a problem and instead the database would be a bottleneck since there are usage limits, although the limits are fairly high. Keeping the database and further developing with it in mind would add unnecessary complexity to our software. The Android application however would have multiple end users making calls at the same time as each user would be making their own calls to the Google Maps and EONET APIs, which was the main reason why we originally proposed a database. On account of the lack of API limits,

there is no need to worry about exceeding the call limit and having to pay for the services.

3.4 Data Collection API

The purpose of this component is to collect natural disaster data raw from various sources and send it to the database. From there the Google Maps API will pull the data and display it for the user on their application. The user will never be directly interacting with this component, neither will the Google Maps API. The only component in the system that interacts with it is the database. The NASA EONET API is our choice for this job [1] It provides continuous data on events happening in the present around the globe, including our natural disasters of interest like earthquakes and wildfires.

4 IMPLEMENTATION

4.1 Data Collection

The events data for the natural disasters are collected from requests made to the EONET API upon 60 second intervals. Each request to the EONET API returns all events data as a series of JSON objects. This data provides the specific details for each disaster, including the id, name, category, date and coordinates.


```
"id": "EONET_5280",
"title": "Iceberg A75",
            "description": null,
"closed": null,
"categories": [
{
"id": "seaLakeIce",
"title": "Sea and Lake Ice"
}
     ],
     "sources": [
{
"id": "NATICE",
"url": "https://usicecenter.gov/pub/Iceberg_Tabular.csv"
}

],
"geometry": [
{
"magnitudeValue": 24.00,
"magnitudeUnit": "NM^2",
"date": "2021-03-26T00:00:00Z",
"type": "Point",
"coordinates": [ -60.49, -74.13 ]
},
{
"magnitudeValue": 18.00,
"magnitudeUnit": "NM^2",
"date": "2021-05-07T00:00:00Z",
"type": "Point",
"coordinates": [ -59.74, -72.12 ]
}
]
```


[1] EONET API. https://eonet.sci.gsfc.nasa.gov/docs/v2.1.

Figure 4: EONET GEOJSON Data: The format in which events are requeseted from the EONET API.

4.2 Data Processing

Following each call to the EONET API, the collection of JSON events is parsed through, and the data for all returned events are stored in an array, according to category. A search is then performed on the array to display a marker on the map for each disaster, based on the corresponding category. Upon searching through the array elements, disasters which belong to categories or date and time ranges filtered out by the user are disregarded and not displayed on the map.

4.3 Map Construction

A map of the world is constructed as the user opens the web page, after requesting the users location information a marker is then dropped onto exactly where the user is. However if the user chooses to deny access then by default the map focuses onto California as this was the original focus point of the project. A default zoom is also set, this is set to make sure that only the area within a short distance of the user is shown. There are also constraints set on the map to prevent users from being able to drop down a pin for a street-view.

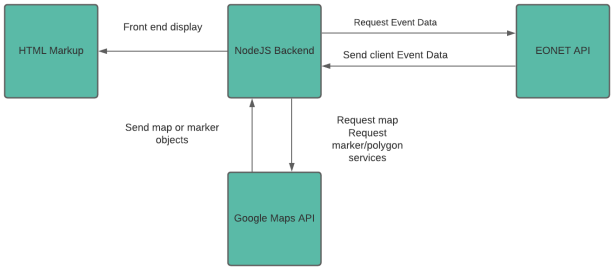

Figure 5: Implementation framework - The final software implementation's component interactions.

4.4 Filtering and Post Construction Processing

After pulling our data from the EONET API, it comes to us in GEOJSON format however all the different types of events are not sorted, so the first thing we do with our data is to sort them depending on their category.

Sorted data is then taken to create markers on our map, before deciding if they should be displayed or not. This is done first because on data refresh, we need to both remove old markers and create the new ones based off of our new data. In order to have a smooth transition between data we remove then re-add markers one category at a time to keep visual downtime as low as possible. Furthermore, for each category we check if the user has enabled the displaying of said category, and if not then we create its mark, and then quickly remove it from the map but store it, in the case that the user decides they want to see it again but do not need to wait for a map refresh.

Once the map is built, markers are placed or in the case of a refresh they are replaced. The user has multiple options for filtering: They can pick between different categories of events via check boxes for each event type. Unchecking one of these boxes will remove all references to it on the map. Checking the box again will display all of the checked events once again. The user may also filter events based on when they began. For example, a user may filter events to only show events which started in March, or events that started in April. Lastly, the user may search by location which will pan the map over to the desired geographical location.

5 RESULTS

In order to test the validity of the information displayed on the map and analyze the response time capabilities, we accessed the site and enabled various filtering capabilities which manipulated the presence of the different disaster markers.

5.1 Experiment #1: Information Display

In our first experiment, we wanted to verify that the map was accurately displaying all real-time disasters pulled from the EONET API. To veryify, the number of open events was calculated by parsing through the length of the JSON object returned by the following request to the EONET API: https://eonet.sci.gsfc.nasa.gov/api/v3/events?status=open. This number was then compared with the count of the number of markers initially displayed on the map. These two numbers were required to be an exact match, in order to validate that the map is accurately displaying the disaster information. The results obtained from this experiment are illustrated by the data in Table 1.

Category	# of Events in JSON	# of Markers on Map
Wildfires	28	28
SeaLakeIce	44	44
Volcanoes	31	31
Overall Total	104	104

Table 1: Experiment #1 Results: Data collected for number of events available in EONET API and displayed on the map

As these results show, all real-time occurring natural disasters were successfully displayed by the map. The remaining 10 disaster categories, drought, earthquakes, floods, landslides, severe storms, temperature extremes, water color, dust haze, man made, and snow, which were available in the EONET API all had 0 open events. These results remained the same, as the map was updated every 60 seconds with new requests to the API.

5.2 Experiment # 2: Filtering Capabilities

In our second experiment, we evaluated the filtering capabilities of the map by collecting the information provided to the user based on requests specific to category, date and time, and location. Since the only real-time disaster data provided were for Wildfires, Sea and Lake Ice, and Volcanoes, these 3 categories were used in our experiment. Several scenarios were taken into consideration.

Firstly, if a category is unchecked from the filter, events corresponding to that category must not be displayed. When all 3 categories are checked, the map must display markers for all 3 events. When only Wildfires and Volcanoes are checked, markers for only those 2 events are to be displayed. If none of these categories are checked, the map must not display any markers. Secondly, if the Date and Time picker is selected, and a start date and time as well as an end date and time is specified, the map must only display the disasters for which the latest event information falls within that range. For instance if a selection of 04/01/2021 12:00 AM - 04/14/2021 11:59 PM is made, the map must only display markers for the open events which have either begun within this time range, or hold information from the API that was last updated within this time range. Finally, if the name of a given location, such as a country, state, town or city, is inputted in the location search box, the map must redirect to to this specified location, and if any disaster events are occurring within that area, they must be shown on the screen. The screenshots presented in Fig. 6, Fig. 7, and Fig. 8 display the different views of the disaster map presented when the filters have been enabled by category, by category and date and time, as well as category, date and time and location, respectively.

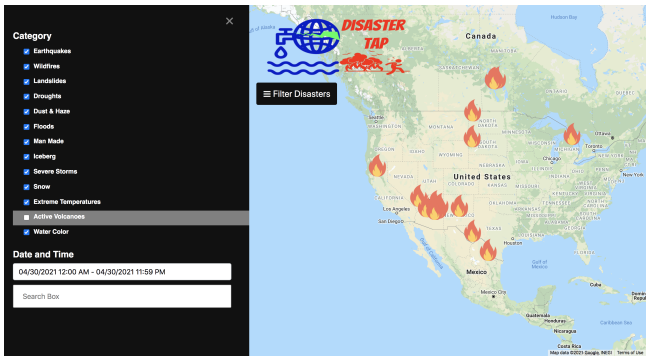

Figure 6: Filter by Category - Display of map with filtering enabled for all Categories except Volcanoes, disabled for Date and Time and Location

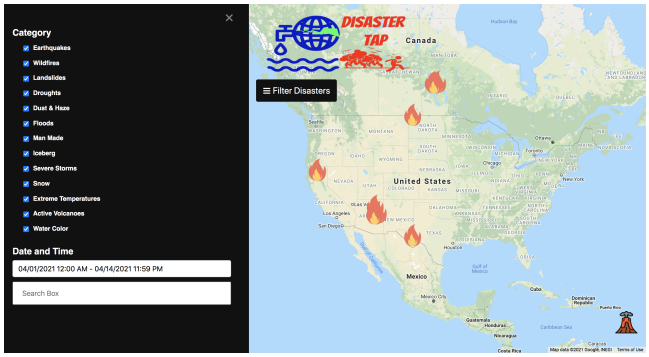

Figure 7: Filter by Category - Display of map with filtering enabled for all Categories, 04/01/2021 12:00 AM - 04/14/2021 11:59 PM Date and Time, disabled for Location

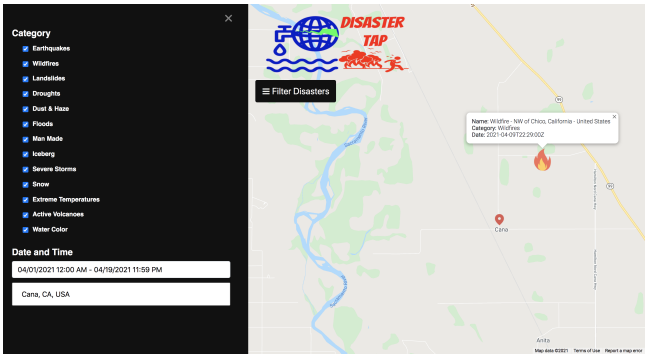

Figure 8: Filter by Category, Date and Time and Location - Display of map with filtering enabled for all Categories, 04/01/2021 12:00 AM - 04/19/2021 11:59 PM Date and Time, and Cana, CA, USA for Location

As these results demonstrate, the map successfully implements filtering requests made by the user. Additionally, multiple filters can be enabled at the same time, providing the user with the option to customize their view of natural disasters or navigate to locations relevant to them. Filters continue to remain enabled as the map was updated every 60 seconds with new requests to the API.

6 CONCLUSION AND FUTURE WORK

In conclusion, our goal was to create a software system to map current, real-time events happening in the world using various different API tools and libraries. At the moment, our software correctly displays various types of world wide events, and is updated while it is being used. Furthermore, the user has various filtering options for time, categorical, and geographical purposes.

Taking this work further may go in different directions. One possible approach would be to use a more complex API in order to track any types of movement an event may have, for example the moment a wildfire begins moving in another directions we may have some indicators of this and notify our users. Another approach to take would be to simply add more features on top of our current build such as when clicking an event will trigger a pull up list of social media accounts from local authorities giving announcements and directives regarding the event or an IoT approach where the website could be linked to a device in the users home to alert them of the disaster information. Similar to what Maswood et al talked about an IoT system for weather notification where users could be notified via a red LED in the shape of an umbrella in approaching storms [9].

ACKNOWLEDGMENTS

We would like to thank Dr. Ani Nahapetian (CS professor at CSUN), as she directed us towards researching the Leaflet API that eventually transitioned to our use of the Google Maps API in our implementation. Dr. Nahapetian had experience with data visualization on maps and mentioned that the Leaflet API is better suited for a data visualization project due to its vast quantity of plugins which enable a more detailed visual. We would also like to thank Professor Xunfei Jiang for her frequent feedback in the midst of development of this project and the providing of strategies and tools that assisted our work in this course as well as towards our future endeavors.

REFERENCES

[1] J. Derrough. 2013. *Instant Interactive Map Designs with Leaflet JavaScript Library How-to.* Packt Publishing. https://books.google.com/books?id=7h9bNGslKZgC
[2] M. Di Giacomo. 2005. MySQL: lessons learned on a digital library. *IEEE Software* 22, 3 (2005), 10–13. https://doi.org/10.1109/MS.2005.71
[3] Michael Erskine and Dawn Gregg. 2012. Utilizing Volunteered Geographic Information to Develop a Real-Time Disaster Mapping Tool: A Prototype and Research Framework.
[4] Shunfu Hu and Ting Dai. 2013. Online map application development using google maps API, SQL database, and ASP.NET. *International Journal of Information and Communication Technology Research* 3 (01 2013).
[5] A. A. Kolesnikov and P. M. Kikin. 2016. DEVELOPMENT OF MAPPING APPLICATIONS FOR MOBILE DEVICES. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLI-B6 (Jun. 2016), 165–168.
[6] Legislative Analyst's Office. 2020. Main Types of Disasters and Associated Trends.
[7] Toshihiro Osaragi, Ikki Niwa, and Noriaki Hirokawa. 2017. *Development of Web Application for Disaster-Information Collection and Its Demonstration Experiment.* 63–73. https://doi.org/10.1007/978-3-319-44711-7_6
[8] Dastgir Pojee, Farooq Shaikh, Mohd Abbas Meghani, and Sajjad Zulphekari. 2018. Intelligent disaster warning and response system with dynamic route selection for evacuation. In *Proceedings of the International Conference on Intelligent Sustainable Systems, ICISS 2017.* Institute of Electrical and Electronics Engineers Inc., 1–5.
[9] Mirza Mohd Shahriar Maswood, Uzzwal Kumar Dey, Md Ashif Uddin, Md Mainul Islam Mamun, Shamima Sultana Sonia, Moriom Akter, and Abdullah G. Alharbi. 2020. A Novel Website Development for Weather Notification System using Smart Umbrella Based on Internet of Things. In *2020 11th IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON).* 0129–0133. https://doi.org/10.1109/IEMCON51383.2020.9284909
[10] Andre Zerger and David Ingle Smith. [n.d.]. *Impediments to using GIS for real-time disaster decision support.* Technical Report. www.elsevier.com/locate/compenvurbsys