College of Engineering
COMP 491 – Computer Engineering Design Project
Final Report

Crowdie: A Crowdsourced News Application

Participant information

Ahmet Akkoç 64741
Deniz Öztürk 64829
Gözde Kahraman 60435
Umut Beşiroğlu 64453

Project Advisor: Öznur Özkasap

Spring 2021

Table of Contents

1. Abstract

Crowdie is a crowdsourced news application where users can report on current social events or emergency situations. The source for the reported events will remain the users, who have to register an account first. Besides base functions of reporting and viewing, users can filter events of their choice, look up events using the navigation functionality and Re-Port events which can be thought of as an event confirmation. Besides the functionalities of our web application directly visible to the user end, privacy and security measurements are also implemented. Sensitive user information is encrypted, and anomaly detection methods are used in order to detect and prevent false information. There also exists a wait time in order to prevent the most obvious method of spam.

2. Introduction

The abundance of knowledge in the age of information initially seemed as a blessing, but it now constricts the reliable communication of news [1][2]. Whether this is due to mal intent or simply lack of knowledge on events; false positives, censorship and manipulation have up a large part of the Internet. The World Wide Web is an interactive medium, in which a large part of trust and security is dependent on its users. In contrast to traditional media, the modern media is now much harder, near impossible, to regulate. As a solution we present Crowdie, a web application aiming to bring out the positives of modern media. After all, the fact that anyone can report any event without relying on the traditional media, albeit its risks, can be a tool to efficiency and integrity during the spread of information [3][4].

Our application aims to be a reliable and efficient information hub, for reporting emergencies, news and social events. With Crowdie we aim to raise awareness on social events that otherwise might have gone unnoticed, and secure efficient responses to honestly reported emergency events. As our source of information will remain the general public, our users, absolute honesty and the lack of any ulterior motive whatsoever simply is not possible. Yet, we supplement this system with a layer of security measures to mitigate such risks. What we have designed is a system which rewards honesty, by giving voice to independently recurrent reports to ensure efficient and reliable knowledge dissemination.

Reportable events are split into a number of categories which were determined through a survey, which had over 800 participants, as well as internal brainstorming. A link to the results of this survey may be found in the Appendix. Having analyzed and discussed what the challenges people faced in trying to access secure and reliable news were, our focus was geared towards local news, where the reports normally wouldn’t be able to gain proper attention in mainstream media. Besides their corresponding numerical value, event categories are color coded as well to provide a

more user-friendly experience. After the implementation of all parts, tests have been performed in order to both ensure the correct workings of anomaly detection and possible bugs that could affect user experience or the integrity of the reports.

In the remainder of this report, Section 3 deals with system organization and implementation details, Section 4 features an analysis of our results and Section 5 provides concluding remarks.

3. System Design

In this section we discuss the system architecture of Crowdie, provided features and finally security mechanisms.

3.1 System Architecture

The devices which are a part of Crowdie form a star network, with Crowdie Viewer clients acting as hosts and the Crowdie Server acting as a central hub. Users run the Crowdie Viewer web application to connect to the server, hosted on Supabase. The server essentially tracks global user and report data. As new events are reported and updates are made, the server sends real-time updates to the other clients on the network. These updates are finally filtered according to the configuration of the user's Crowdie Viewer instance.

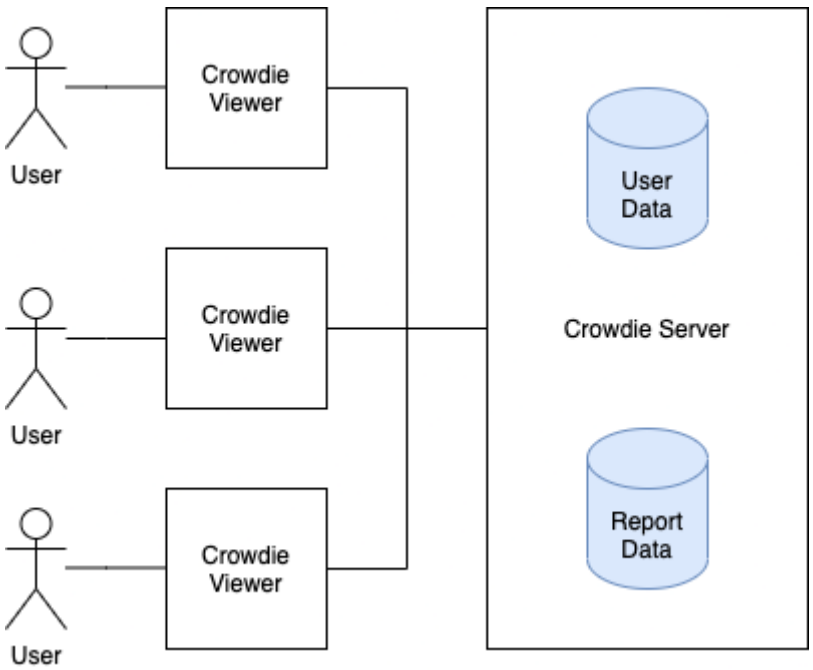


Figure 1: Crowdie System Architecture

3.1.1 Crowdie Viewer

The Crowdie Viewer interface allows users to follow reported events and to report new events of their own. or approve the event reports around them. Our interface was coded in React

Native and runs on a Node.js server.

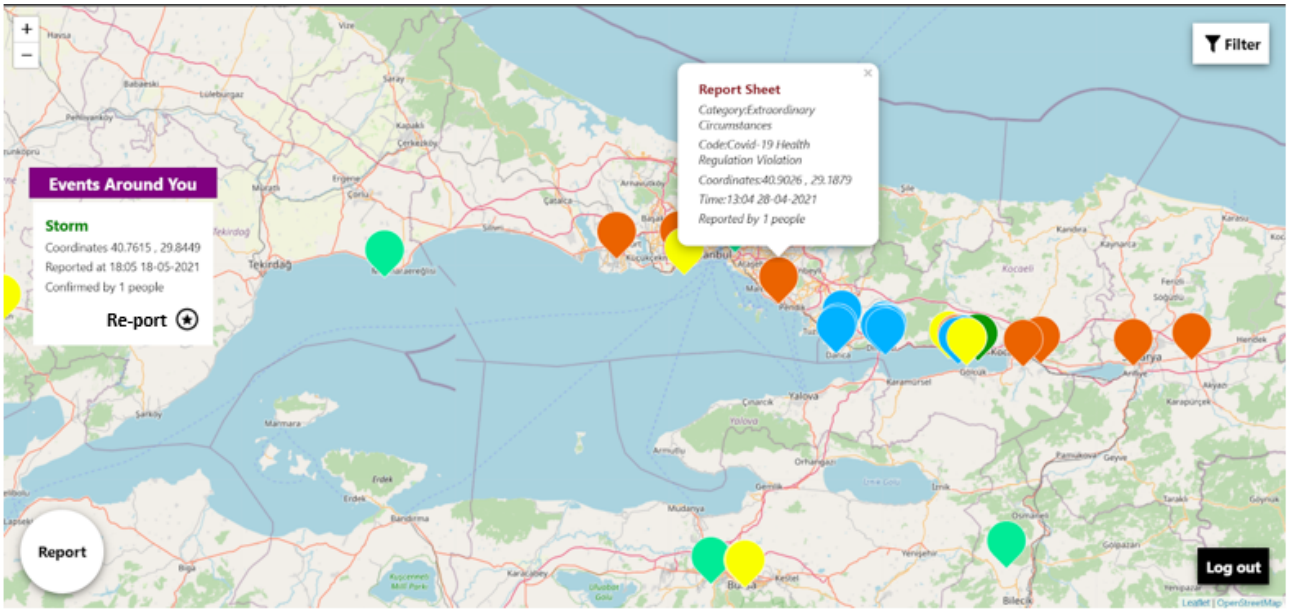


Figure 2: Crowdie Viewer

The Viewer takes the form of pins layered on top of Open Street Map. Each of these pins represents a particular event report (or group of reports). These are color-coded, based on category and can be clicked on to view a more detailed explanation. Any user can view events as they please, but reporting events requires the user to register and log in.

On the left side of the screen is the *Events Around You* panel, which provides a quick summary of nearby events. From here, users can *Re-Port* (short for REpeated rePORT) events which they would like to confirm. To the lower-left corner is the Report button for reporting new events. Opposite of it in the upper-right of the screen is the Filter button whose function is described in section 3.2.3.

3.1.2 Supabase

Supabase (Supabase Contributors 2020) is a hosting platform which provides PostgreSQL Databases, user authentication, real-time subscriptions. This configurability attracted us to host Crowdie's server-side backend operations on Supabase.

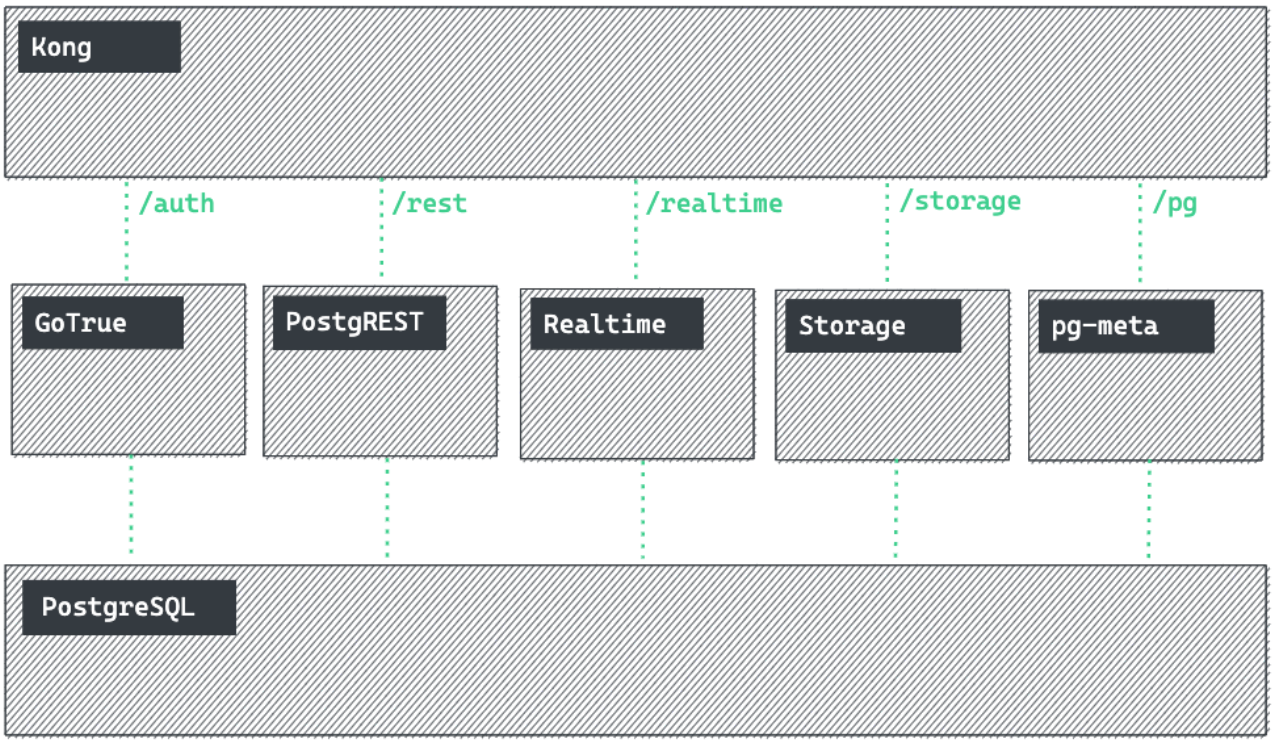


Figure 3: Components of Supabase (Supabase Contributors 2020)

To track user reports we keep a report table where for each report we store its type (see Table 1 in Section 3.3.2), its coordinates and the time it was made. As new reports are made to the system, Supabase pushes these updates to the Crowdie Viewer clients automatically.

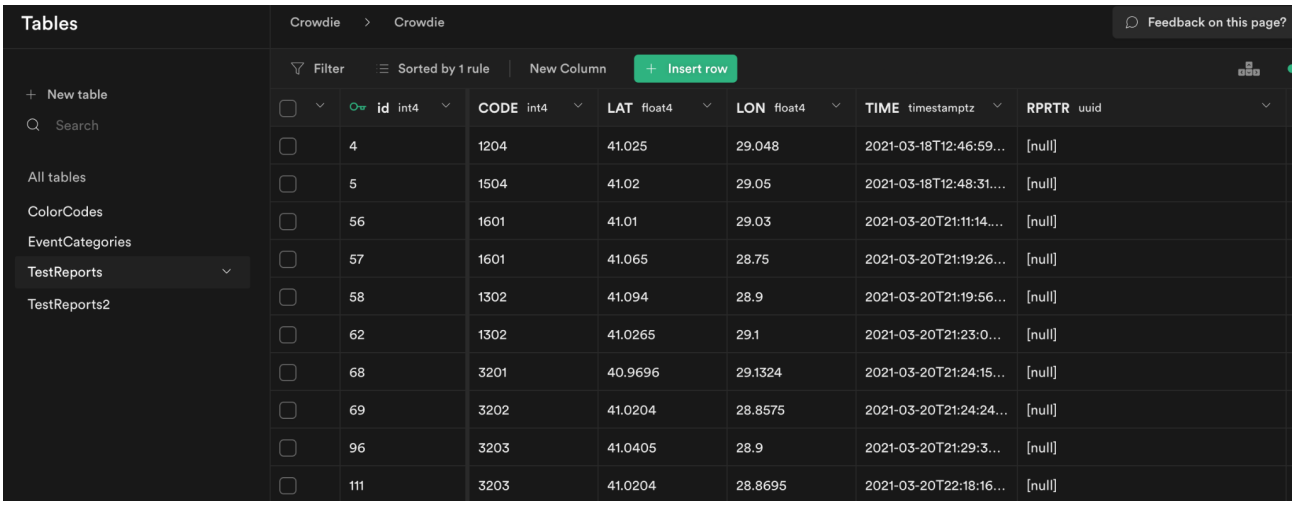


Figure 4: Components of Supabase (Supabase Contributors 2020)

3.2 Features

Crowdie users fit into one of two roles: readers and reporters. By default the user is a reader, Crowdie loads the map around their geolocation. The reader can scroll around to reveal more of the map or view a summary in the *Events Around You* panel. The reader can also zoom out to see events

in other parts of the world, or filter by category. In fact, the reader can even set the time frame they are looking at, instead of being limited to only the last 24 hours.

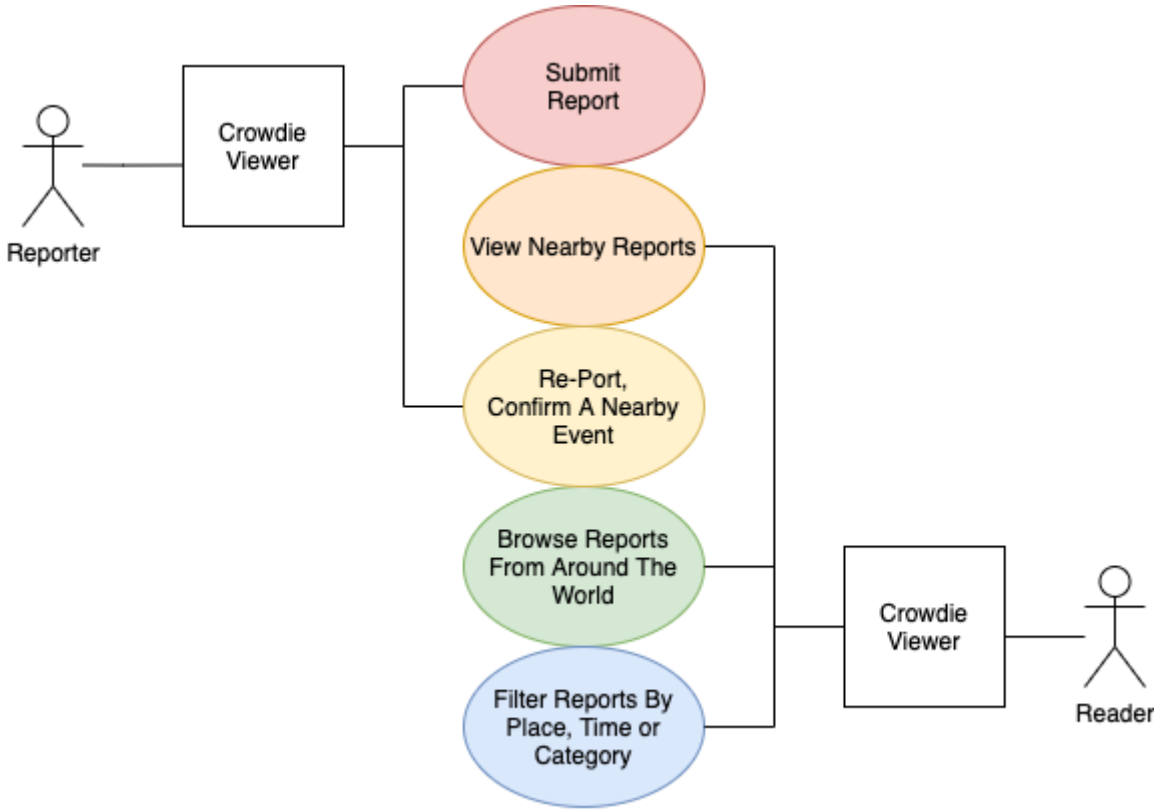


Figure 5: Crowdie Use Case Diagram

Registered users gain access to report privileges. A reporter can report an event around them, from a list of ~30 events. A reporter can also Re-Port nearby events to confirm that they have also seen or witnessed a particular phenomena. It should be noted that event reports are anonymous, that is to say no particular report is associated with its reporter, further details on the degree of this anonymity can be found in Section 3.3. Registration is only required to be granted report privileges, no further data is collected from registered users.

3.2.1 Navigation

Navigation is an important part of our project as it is the main feature for the viewer. Our goal in receiving and displaying the reports was to receive each report separately and display them separately based on three criteria: Geolocation data, timestamp and report code. The Javascript API we used for geolocation of users also allows the current timestamp to be added as an attribute. This API is implemented inside of a component called 'Nav.js' where noise is added to the geolocation coordinates received from it using Planar Laplace which will be elaborated on in Section 3.3.2.

```
Longitude is : 29.0489544664767
Latitude is : 41.191582885088174
Timestamp is : 1623191567378
```

```
Longitude is : 29.049000970595056
Latitude is : 41.191593207393225
Timestamp is : 1623213558208
```

Figure 6: Geolocation outputs at different times from the same place

As can be seen in Figure 6, the longitude and the latitude have changed slightly even though both of these geolocation snapshots were taken from the same device, at different times. After this data is generated on the client side, geolocation data is combined with the report data and sent to the server which adds the report to the reports table. Finally, all of the reports are later fetched to be displayed on the clients' map. We have used React Leaflet as our map API which uses OpenStreetMap as its basis and provides necessary tools for our purposes such as markers where report descriptions are listed.

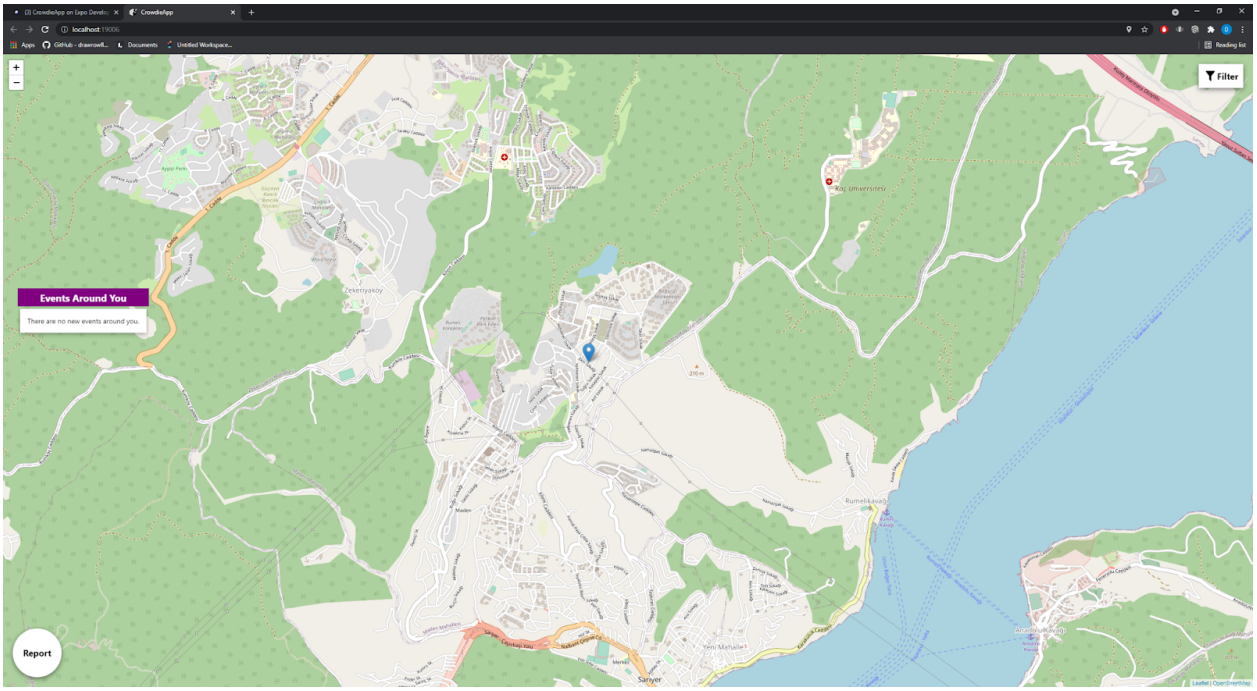


Figure 7: Location marker that shows the user's location on the map

As shown in Figure 7, the marker is started at the user's location after the user gives the necessary consent for their location data to be used. Since this data is indispensable for our application, we have decided that users who don't give consent for their location to be seen and used, will not be able to report events; although they may still continue to browse reports given by other users.

3.2.2 Reporting

As stated above, event reporting is a feature only available to registered users. We used the results of our research survey of over 800 people to identify reportable event options. As a result of

the survey analysis, we gathered 31 events under 10 main categories:

Main Category	Event	Main Category	Event
Generic Infrastructure Problem	Water Outage	Generic Environmental Problem	Litter
Generic Infrastructure Problem	Natural Gas Outage	Generic Environmental Problem	Clean-water Pollution
Generic Infrastructure Problem	Electricity Outage	Generic Environmental Problem	Air Pollution
Generic Infrastructure Problem	Internet Outage	Generic Health Problem	Unable to receive medical services
Generic Traffic and Transportation Problem	Roadwork	Generic Health Problem	Unable to receive vaccination
Generic Traffic and Transportation Problem	Traffic Accident	Generic Health Problem	Unable to receive treatment
Generic Traffic and Transportation Problem	Lights are broken	Extraordinary Circumstances	Unable to receive Covid-19 testing
Generic Traffic and Transportation Problem	Traffic Lights are broken	Extraordinary Circumstances	Unable to receive Covid-19 vaccination
Natural Disaster	Flood	Extraordinary Circumstances	Covid-19 Health Regulation Violation

Natural Disaster	Storm	One-day Social Events	Concert
Natural Disaster	Earthquake	One-day Social Events	Play
Generic Urban Problem	Sudden Bang	One-day Social Events	Bazaar
Generic Urban Problem	Noise Pollution	Daily Social Events	Fair
Generic Urban Problem	Environmental Odors	Daily Social Events	Festival
Generic Urban Problem	Celebratory Gunfire	Daily Social Events	Art Gallery
Spontaneous Social Events	Flash Mob		

Table 1: Events and Categories

When users want to share an event happening around them, they select the most appropriate category and report the event. To protect the privacy of the user, the identity information of the person reporting the incident is not collected, they are only required to be logged in. In the report information, only the information of how many people reported the event is kept. After the report is received, it goes through minor processing as part of our security measures, as described in Section 3.3.

Instead of reporting a new event, users can also confirm previously reported events that have happened around them. Not every event shown on the map is eligible for confirmation. We assume that the person may have been aware of a recent event in their environment, so only events that appear within a certain radius of the user can be confirmed. We apply a filtering on the events to determine events around the user. We draw an invisible square around the user and list these under the Events Around You section, showing the events inside this region which were reported in the last twenty four hour. Users can confirm events around them by clicking Re-Port. When the number of confirmations increases, the marker of that event will turn more opaque on the map.

3.2.3 Event Filtering

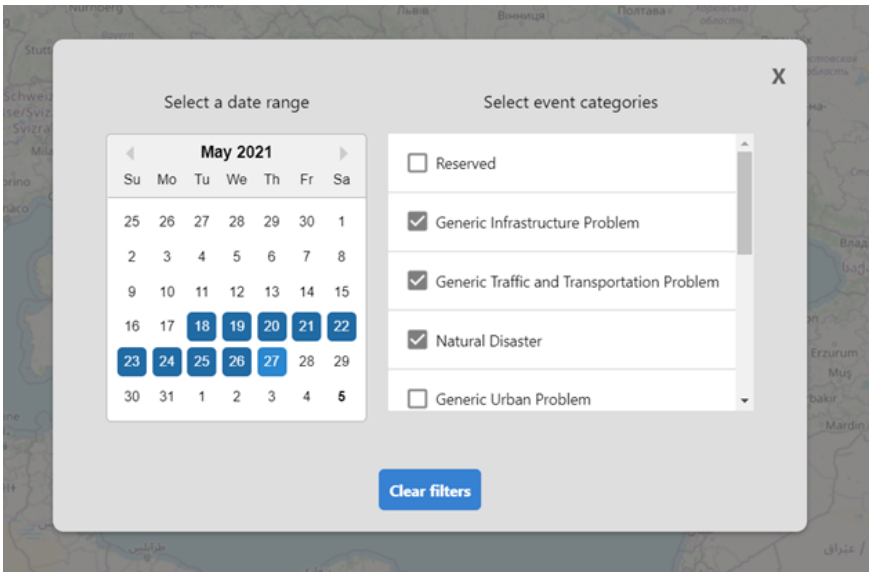


Figure 8: Filtering Options

By default, all event reports reported in the last 24 hours are displayed on the map. We assume that the number of users and the frequency of event reporting will increase in the future, which may make it difficult for users to find a particular event on the map. To eliminate this difficulty, we have added a filtering feature to the application.

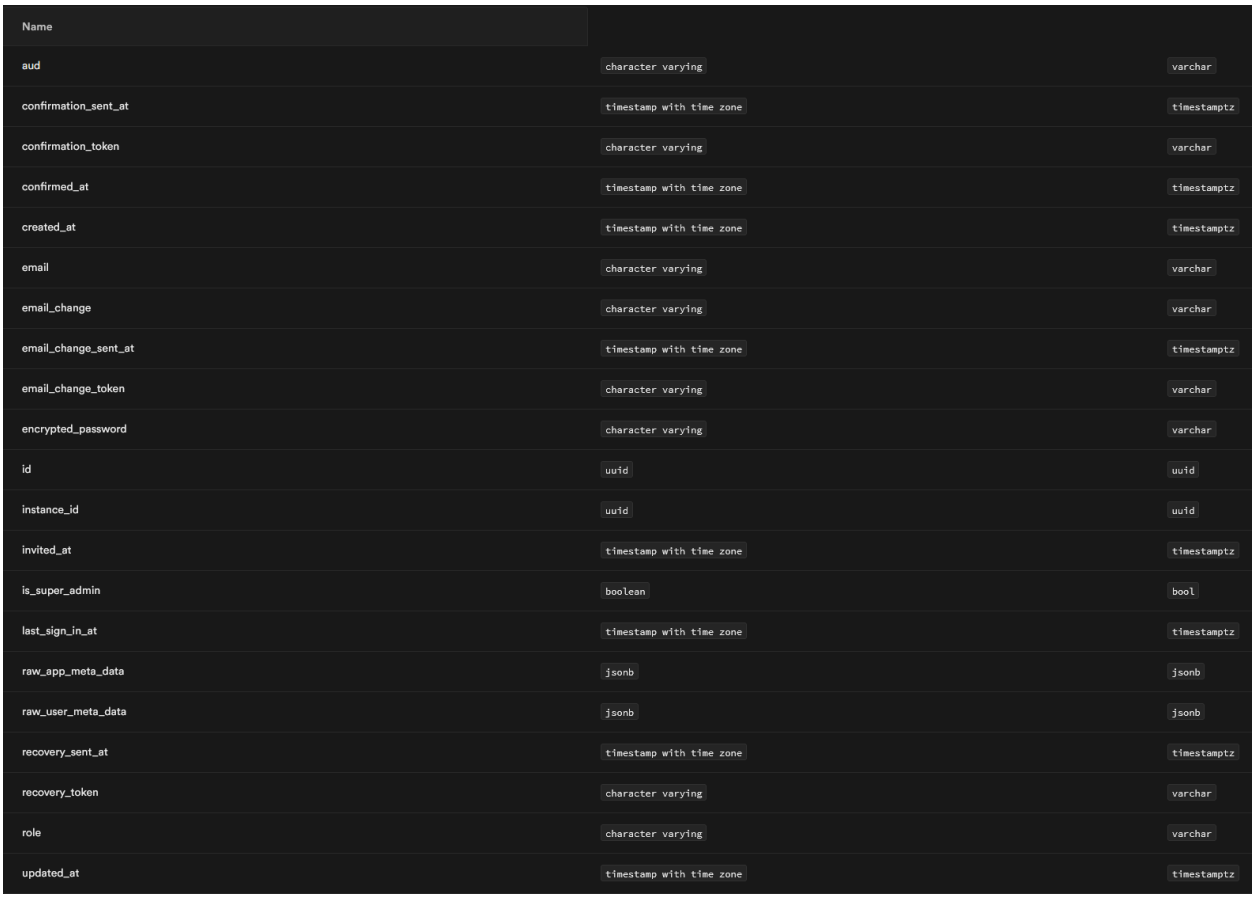
There are two filtering options, you can select a time range or select event categories. If you want to remove these filters you can remove them by clicking the clear filters button. When the filters are cleared, the system continues to display the events reported in the last 24 hours, which is the default filter.

3.3 Security Measures

Our main security concerns for the project were: maintaining user privacy, preventing spam and mitigating false reports. Our first intention after research[5] was to use a Blockchain since it solved all three of our problems but it brought another huge issue: Consensus. The amount of computing power required to add a single report to the ledger was too high to be maintainable or scale. Even using a permissioned blockchains didn't seem efficient enough for our security issues. Therefore we decided to tackle each issue independently.

3.3.1 Account Protection

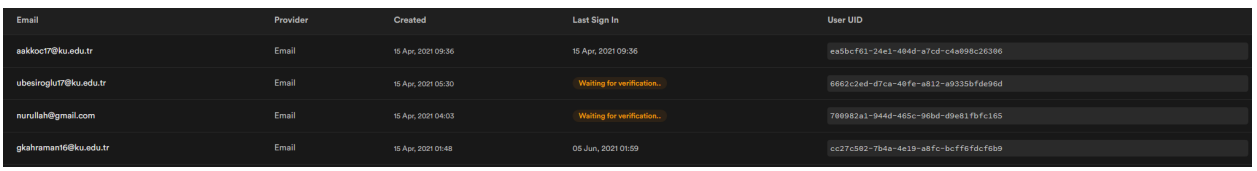
Since users have to register and sign in to use the report feature, their account information needs to be secure and no other user should be able to access them. Supabase provides an ‘Authentication Schema’ where a table for each user is created which contains all of the information related to that user such as email address, id, last sign-in, role and most importantly encrypted password; as can be seen in Figure 9.



Name		
aud	character varying	varchar
confirmation_sent_at	timestamp with time zone	timestamptz
confirmation_token	character varying	varchar
confirmed_at	timestamp with time zone	timestamptz
created_at	timestamp with time zone	timestamptz
email	character varying	varchar
email_change	character varying	varchar
email_change_sent_at	timestamp with time zone	timestamptz
email_change_token	character varying	varchar
encrypted_password	character varying	varchar
id	uuid	uuid
instance_id	uuid	uuid
invited_at	timestamp with time zone	timestamptz
is_super_admin	boolean	bool
last_sign_in_at	timestamp with time zone	timestamptz
raw_app_meta_data	jsonb	jsonb
raw_user_meta_data	jsonb	jsonb
recovery_sent_at	timestamp with time zone	timestamptz
recovery_token	character varying	varchar
role	character varying	varchar
updated_at	timestamp with time zone	timestamptz

Figure 9: User table in Supabase

Therefore even on the backend side, we cannot view users' passwords. The only information available to us are: email, email provider, account creation date, last sign-in date and user-id which is shown in Figure 10. Also, since we use Linear Differential Privacy (to be discussed in the next subsection) to add noise to a user's geolocation data, if an attacker were to view our tables, they wouldn’t be able to pin a user’s exact location.



Email	Provider	Created	Last Sign In	User UID
aakkoct7@ku.edu.tr	Email	15 Apr, 2021 09:36	15 Apr, 2021 09:36	ea5bcf61-24e1-404d-a7cd-c4a898c26386
ubesiroglut7@ku.edu.tr	Email	15 Apr, 2021 05:30	Waiting for verification...	6662c2ed-d7ca-40fe-a812-a9335bfde96d
nurullah@gmail.com	Email	15 Apr, 2021 04:03	Waiting for verification...	700982a1-944d-465c-96bd-d9e81fbfc165
gkahramant6@ku.edu.tr	Email	15 Apr, 2021 01:48	06 Jun, 2021 01:59	cc27c582-7b4a-4e19-a8fc-bcff6fdcf6b9

Figure 10: Filled-in user table in Supabase

3.3.2 Local Differential Privacy

As mentioned before, user privacy is one of the most important issues for our project and the most revealing data was their geographical location data which is very valuable on its own. As a solution we turned to local differential privacy. In line with our research [6] we implemented a Planar Laplace Mechanism, adapted from the 'location-guard'[7].

This component first converts the longitude and latitude received from the user to radian using earth's radius. Then, using the inverse cumulative polar laplacian distribution function, adds a noise using a predetermined number 'epsilon' to the position (in radians) while ensuring that the new position is with high likelihood within a bounded distance from the original position. In effect, every report is slightly different from reality, making the report un-associable with the reporter's real address[8]. On the other hand, we expect the noise added in this manner to cancel out in aggregate results, meaning that if we have a dense cluster of similar reports, said cluster will reflect an event concentrated in that particular region.

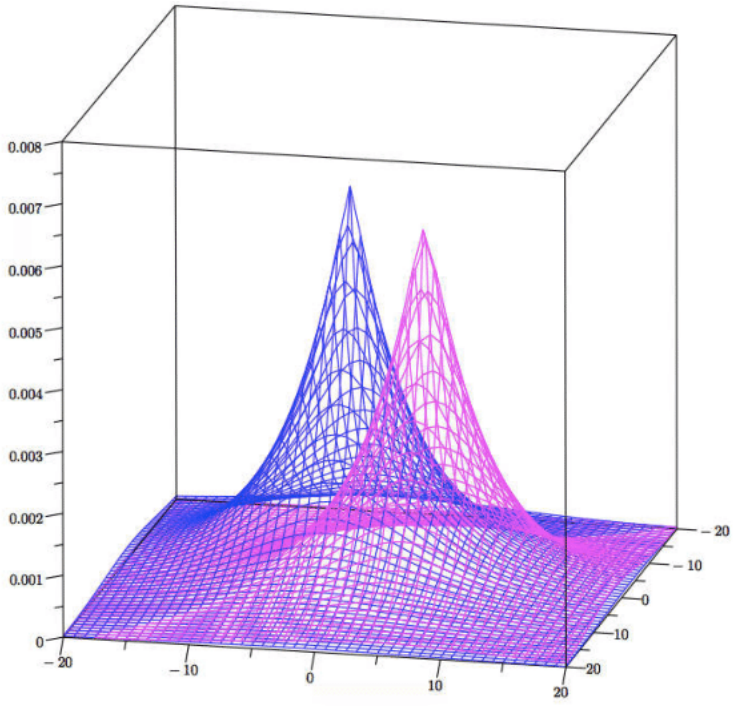


Figure 11: Planar Laplace Distribution[8], points closer to the center are likelier to be sampled than points further from the center.

In our implementation, the Crowdie Viewer program will request the user's geolocation *locally* and perturbate it with noise (again locally) before reporting to the server. Thus, the user's real location is never shared with the server.

3.3.3 Anomaly Detection

Another important issue is false reporting. Since our app is geared towards emergency situations, false reports could harm others and may cause responses to be provided incorrectly or unnecessary civil distress. Based on previous discussion, we noted that one way to verify the authenticity of a reported event, is to see if we can get independent recurrent reports. At first we thought of having a secondary storage, which would release events when multiple “blind”-reporters had submitted a certain number of reports. Later we saw that we could achieve a similar effect by utilizing anomaly detection techniques from machine learning.

We decided to prevent false reports from displaying on the map using isolation forests, as implemented by the npm package ‘isolation-forest’[10]. This implementation takes as input a ‘number of trees’ to score each report, under the current filter settings, based on its geographical location and (UNIX) timestamp, from 0 to 1. All of the reports were scored with respect to other reports of the same code to ensure the elimination of legitimate reports of certain types with lower reporting rates. If the score of the report is close to 1 it is considered an anomaly and removed from the list. If it is less than 0.5 it is regular and stays in the list.

Using this library we have managed to prevent singular reports that weren’t close to the cluster of reports of the same code from displaying on the map and ensured that fake reports are hard to manufacture with limited resources.

3.3.4 Wait Time

One last issue is report spamming. If users were able to spam reports, they could create events that seem legitimate since anomaly detection wouldn’t be able to pick up on them. Hence we implemented a cooldown time where each user can only report events once every 5 minutes.

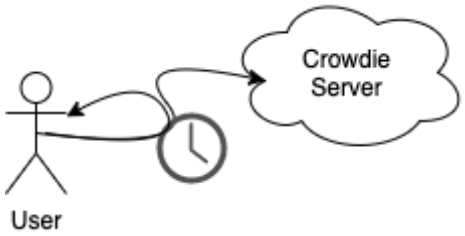


Figure 12: Wait Time Logic

4. Analysis and Results

Our goal for navigation was to enable users to view and report using a map and limit their reports to their geoposition for reliability’s sake. By implementing a map and its components with React Leaflet, we have built a report viewer overlayed on a map where markers with different

colors represent different events and each of them have descriptions such as date, category, number of reports etc. of the event. Hence, users can view reported events and relevant information by navigating the map which is simpler than looking through a list as users will be interactting with visual elements.

Security solutions for our project were controversial. Our first idea was using blockchain technology but as mentioned before, consensus was a tremendous issue for a simple application like ours and therefore wasn't scalable since that would require enormous amounts of computing from nodes. It provided solutions for spam, account security and false reporting with the usage of consensus and validation of each report, but because of the issues with scalability and a need for high computation we have decided on a simpler solution which is to solve each issue separately. Supabase provided account security with password encryption and we have implemented Linear Differential Privacy to hide users' exact locations. Also, by implementing anomaly detection with 'isolation-forest', we have solved the issue of false reporting since the isolation forest algorithm partially eliminates abnormal data points in a set. Lastly we have implemented 'wait time' for users to wait after submitting a report, preventing them from spam reporting. With all these features, security issues we aimed to solve with blockchain at the start were solved but in a centralized manner as opposed to being distributed. Hence, still a security issue remains: Supabase being hacked. Using blockchain would be more secure in that regard since it is a distributed ledger. It would still be prone to attacks since there is encrypted information on there as well. But as opposed to attacks being directed to a central server, they would be directed towards users to obtain their private keys that would be used to encrypt their information.

Initially, our goal was to show each event as a bubble on the map and increase the size of the bubble as the number of event reporting increases. In order to do this, we had to give the reports of each event group as an input to a clustering algorithm, create event clusters, and show each cluster on the map with a bubble with a diameter proportional to the number of elements in the cluster. However, when we examined the existing map marker clustering libraries, we could not find a suitable library for our needs. This was mainly due to geospatial libraries not really supporting a third dimension, while we were also working with temporal dimensionality.

Existing libraries do clustering based on either coordinates or time. We considered using the two algorithms one after the other, but in this case, re-running the algorithm every time an event is added and updating the map would not be a very efficient solution in terms of performance, so we developed an alternative solution. Each user can report a previously undeclared type of event in his/her environment, and can also confirm previously reported events. Thus, events are still, in a sense, clustered on their first reported locations. As the number of confirmations increases, the

visibility of events on the map increases. Thus, events that have been approved by fewer people are shown with a more transparent marker, and events that have been approved by more people are shown with a more opaque marker.

5. Conclusion

We have designed Crowdie as a crowdsourced application because we believe that traditional media or even social media fall short where emergency arises. We hope that what we have designed over the past few months can be extended to make news viewing and, more importantly, reporting accessible to the majority of the population. It was our goal to fill this void and help decrease the response time to emergencies and make a larger crowd aware of them. Therefore we designed Crowdie as a simple React project with a basic Graphical User Interface and used a myriad of libraries to enhance the functionality. By using these libraries and our own design together we ensured the security of users and the accuracy of the data viewed by them.

In the end, we achieved the majority of our initial goals with Crowdie and in some aspects surpassed them, such as the reporting system and report categorization. During the project we researched blockchain technology to solve security issues but went along with a simpler solution of separately tackling each security issue. Also we utilized many Javascript libraries to provide a better design for the project and used Supabase for a more robust and secure backend. After the deadline for the project we continued to fix bugs and assure stability of our project and we hope to continue to do so while adding some new features such as clustering, for detecting trends, and GUI improvements.

6. References

[1] Earth Journalism Network. 2016. Reporting on Disasters. [online] Available: https://earthjournalism.net/resources/reporting-on-disasters [Accessed 04-March-2021].

[2] Earth Journalism Network. 2016. Using Social Media. [online] Available: https://earthjournalism.net/resources/using-social-media [Accessed 04-Mar-2021].

[3] Humanitarian OpenStreetMap Team: Home," Humanitarian OpenStreetMap Team | Home. [Online]. Available: https://www.hotosm.org/. [Accessed: 04-Mar-2021].

[4] Crisis Cleanup. [Online]. Available: https://www.crisiscleanup.org/about. [Accessed: 04-Mar-2021].

[5] Crowdie-Project/Research-Logs, *GitHub*, 2021. [Online]. Available: https://github.com/Crowdie-Project/Research-Logs/blob/main/deniz.md [Accessed: 01- Jun- 2021].

[6] Crowdie-Project/Research-Logs, *GitHub*, 2021. [Online]. Available: https://github.com/Crowdie-Project/Research-Logs/blob/main/ahmet.md. [Accessed: 01- Jun- 2021].

[7] chatziko/location-guard, *GitHub*, 2021. [Online]. Available: https://github.com/chatziko/location-guard. [Accessed: 01- Jun- 2021].

[8] Miguel E. Andrés, Nicolás E. Bordenabe, Konstantinos Chatzikokolakis, and Catuscia Palamidessi. 2013. Geo-indistinguishability: differential privacy for location-based systems. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security* (*CCS '13*). Association for Computing Machinery, New York, NY, USA, 901–914. DOI:https://doi.org/10.1145/2508859.2516735

[9] "isolation-forest", npm, 2021. [Online]. Available: https://www.npmjs.com/package/isolation-forest. [Accessed: 01- Jun- 2021].

7. Appendix

Project Document Deliverables:
https://drive.google.com/drive/u/2/folders/1AwAsTvEIsrIsn8AmSsLGbdVcIt88zhZV
Github Repository & Survey Data & Research Logs:
https://github.com/Crowdie-Project/
https://github.com/Crowdie-Project/Survey-Data
https://github.com/Crowdie-Project/Research-Logs

https://docs.google.com/forms/d/e/1FAIpQLSf3MMhRWRduZ7dkZM5Qxo1-I0geGQ0b3Ur7GcK1CVrtxkpSog/viewform