# FloodWatch: A Mobile Application for Real Time Flood Response and Analysis

A Technical Report submitted to the Department of Computer Science

Presented to the Faculty of the School of Engineering and Applied Science

University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree

Bachelor of Science, School of Engineering

**Adriel Kim**

Spring, 2022

Technical Project Team Members

Ankit Gupta

Christian Jung

Preston Wright

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

N. Rich Nguyen, Department of Computer Science

# FloodWatch: A Mobile Application for Real Time Flood Response and Analysis

CS4980 Capstone Report, 2022

Adriel Kim
Computer Science
University of Virginia
School of Engineering and Applied Science
Charlottesville, Virginia USA
ak6fnm@virginia.edu

***Abstract* – FloodWatch is a customizable mobile application that enables its users to access real-time and historical flooding data to enable fast disaster response and data analysis. Currently, the app aims enable residents of Ho Chi Minh City (HCMC) to tackle the regular occurrence of flooding in the city. Through the app, users can upload real-time reports of the current flooding situation in their area. These reports provide an image, estimated severity level, and description of the flood. Other users can then view these reports in real-time to navigate the flooded area. In addition to user-generated data, users can view city weather data scraped from various weather APIs. This app provides an immediate benefit to city-residents while also serving as a means to gather large amounts of flood data for future research.**

## 1 INTRODUCTION

### 1.1. Background

Coastal flooding is a major threat to cities around the world due to factors such as sea-level rise and land subsidence [1]. It is predicted that by the 2050s, "800 million people will live in cities where sea levels could rise by more than half a meter" [1]. Although this is a global problem, this project focuses on Ho Chi Minh City (HCMC), which is among the top ten cities most vulnerable to flooding [2]. Existing flood solutions are costly to implement, have potential negative environmental consequences, and do not provide a long-term solution. For example, the proposed sea dike construction surrounding HCMC is predicted to cost US$3.3 billion [3]. In addition, studies have been conducted that expose the negative environmental impacts of the construction such as a reduction of salinity of the surrounding mangrove forests [4].

An alternative, non-invasive solution is needed that provides both immediate and long-term benefit and improves our understanding of coastal flooding. Ideally, this type of solution would allow city-residents to adapt to flooding rather than resist it through costly methods. In the long-term, such a solution would allow decision makers to determine the most appropriate flood solution or enable future researchers to develop predictive flooding models.

### 1.2. Prior Art

The FloodWatch app was largely inspired by various existing applications including HSDC Maps, UDI Maps, and Risk Factor. HSDC Maps, developed by the Hanoi Sewage and Drainage Company, is a mobile flood warning application for residents of Hanoi. The app provides live flood and rain data; it also provides live feed from 31 different cameras and built-in GPS navigation [5]. UDI Maps is very similar to HSDC Maps, but it was made for HCMC by the Ho Chi Minh City Urban Drainage Company [6].

Risk Factor is an online tool, created by the nonprofit First Street Foundation, for assessing a property's risk of flooding in the U.S. [7]. The tool provides an aesthetically pleasing and detailed report for a property's flood risk based on a location's geographic information such as elevation, climate, proximity to water, etc. [7].

FloodWatch combined the best aspects of each of these applications to address each of their shortcomings. Both HSDC Maps and UDI Maps lack a graphical means to view historical data. In addition, they are only focused on a specific city while FloodWatch aims to be generalized for all flood prone areas. However, their strength lies in their map user interface and real-time user flood reports, which we implemented for FloodWatch. Risk Factor is more suited for long-term flooding forecasts; however, FloodWatch adopted some of its features as well. For instance, FloodWatch adopted the same five-color scheme for visualizing flood severity that Risk Factor uses for visualizing flood risk level. In addition, Risk Factor has inspired FloodWatch's future plans to develop a machine learning model for flood forecasting.

Like HSDC Maps and UDI Maps, FloodWatch provides immediate benefit to its users, allowing them to navigate floods in real-time. And, like Risk Factor, it adopts an interface more suited for analysis and research. What makes FloodWatch unique is its ability to display a range of historical data through a range slider component.

## 2 Materials and Methods

### 2.1. Materials

FloodWatch was developed with a variety of technologies that enabled planning, frontend and API development, data storage, and web hosting. The following are the primary technologies used to develop FloodWatch: ReactJS, Mapbox GL JS, AWS Chalice, Amazon DynamoDB, Netlify, Figma, and Draw.io. The programming languages used to develop FloodWatch are Python and JavaScript.

ReactJS and Mapbox GL JS constitutes the frontend stack of the app. ReactJS is a JavaScript library for developing user interfaces [8]. This was used in conjunction with Tailwind CSS, an inline CSS styling framework [9]. Mapbox GL JS is arguably the most critical library, enabling FloodWatch to display map data in a performant, interactive, and customizable manner [10].

AWS Chalice and DynamoDB makes up FloodWatch's API and backend stack. AWS Chalice is a framework for developing serverless applications [11]. We used this framework to develop FloodWatch's API, which fetches and uploads flood data displayed by the app. Amazon DynamoDB is a NoSQL database service which we used to store the flood data [12]. The AWS Chalice API interacts with DynamoDB to fetch and upload data.

Netlify is a web development platform that provides web hosting [13]. FloodWatch is currently accessible through the web and hosted on Netlify. Figma is a collaborative design tool for user interfaces [14]. Like Figma, Draw.io is a collaborative design tool but for diagrams instead [15]. Figma and Draw.io were critical to our initial planning stages and developing the vision and architecture for the app.

### 2.2. Methods

The process of developing FloodWatch consisted of three main stages: planning, mockup designs, and implementation. The planning stage involved defining the problem FloodWatch is to address, listing the necessary requirements, and establishing the app's competitive advantage over existing solutions. After that, we developed mockups to establish to establish a vision shared between team members and collaborators. These mockups were then used to guide implementation of the app. We used a feature-based workflow, writing the necessary code at all levels of the software stack to implement a certain feature.

Defining the problem definition involved background research into the problem of flooding in HCMC. This research served as motivation for the project and allowed us to establish its goals. This research informed our features and user interface (UI) design decisions. Based on our understanding of the problem, goals for the project, and inspiration from prior art, we developed a list of the necessary requirements and features.

Based on this list of requirements and features, we then developed UI mockups and an architecture diagram of the FloodWatch app [Fig. 1]. These mockups and diagram would serve as a guide for the implementation of the app.

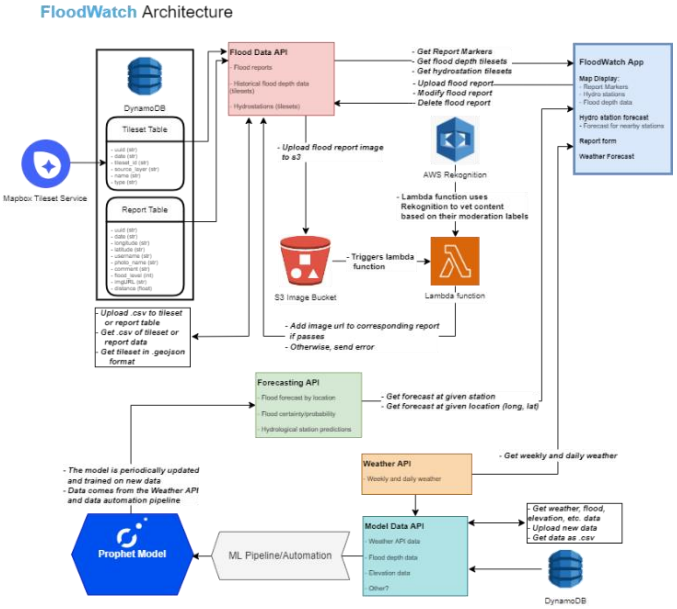


Fig. 1. FloodWatch architecture diagram

Once planning was complete, we implemented the app using a feature-based workflow. For example, one of the first features implemented was the flood report upload form. This form allows users to upload a flood report consisting of an image, an estimated flood level, and a description to a DynamoDB database. Implementing this feature involved both frontend and backend development. Of course, developing the frontend entailed creating the user interface using React and Tailwind CSS. And the backend required developing an API using AWS Chalice and DynamoDB in order to upload and store data. A similar process was used to display the report data on the map, enable the reports to update in real-time, and allow the user to scroll through historical data.

## 3 Results

FloodWatch is a functioning platform for uploading and displaying flood data on a map in real-time. In addition, it has core features such as a range slider to scroll through historical data, an options menu to customize the map and filter the data, and a weather information widget. Since FloodWatch is a progressive web app (PWA), it is accessible on both the web and as an installable mobile app. PWAs are basically web apps but with native app features such as the ability to be installed, work offline, send push notifications, etc. [16].

When a user wants to upload a flood report, they can navigate to the warning triangle icon to fill out the flood report upload form previously described. The user then can upload an image of the flooding situation in their area. They can also add a description and choose one of five flood levels which are based on flood depth [Fig. 2]. Once the user presses submit, the map is updated automatically via a Web Socket API we created using Amazon's API Gateway. This flood

report appears as a marker on the map; the color of this marker corresponds to the flood level. Users can press this marker to view more details [Fig. 3].

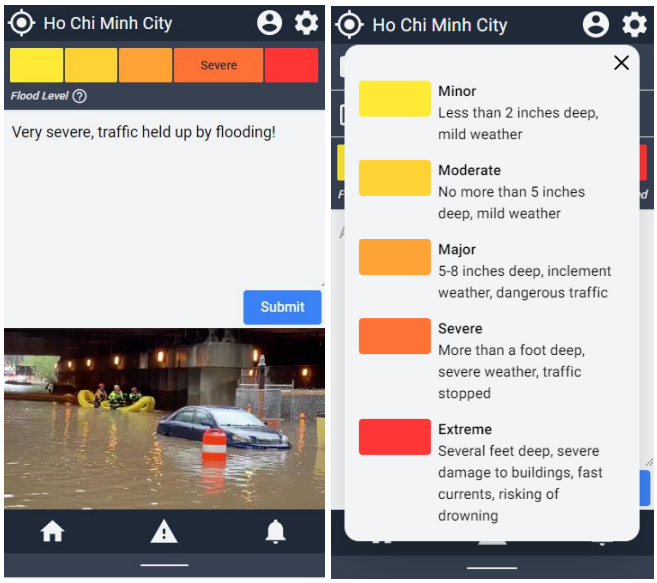


Fig. 2. Flood report upload form (left) and flood level descriptions (right)

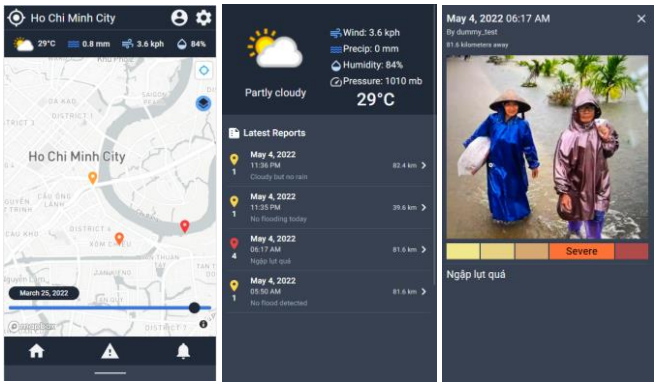


Fig. 3. Flood map view with range slider (left), flood report list view (middle), and flood report detailed view (right)

The user can customize the flood map by pressing the blue bubble icon on the top right of the screen. This opens up a menu of options, allowing the user to change the map's style, change the visibility of different types of data, and filter out data by flood level [Fig. 4].

Besides user generated flood reports, FloodWatch also displays historical city weather data [Fig. 5]. The user can adjust a range slider to visualize how this data changes over time. The points on the map are colored based on precipitation level. The attributes provided are humidity, precipitation (mm/h), temperature (°F.), and wind speed (m/s).

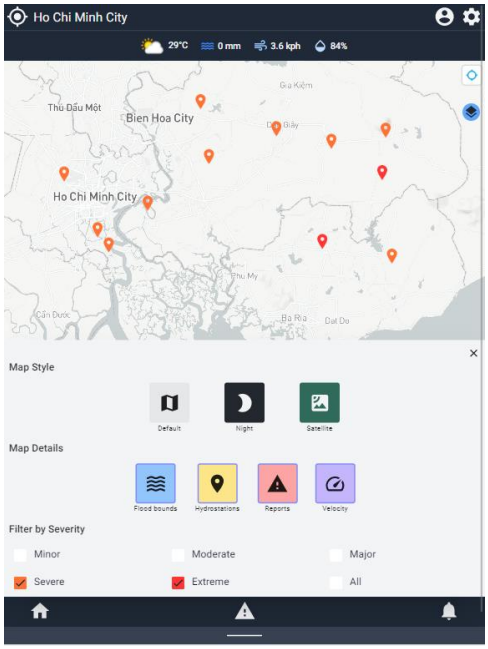


Fig. 4. Flood map options

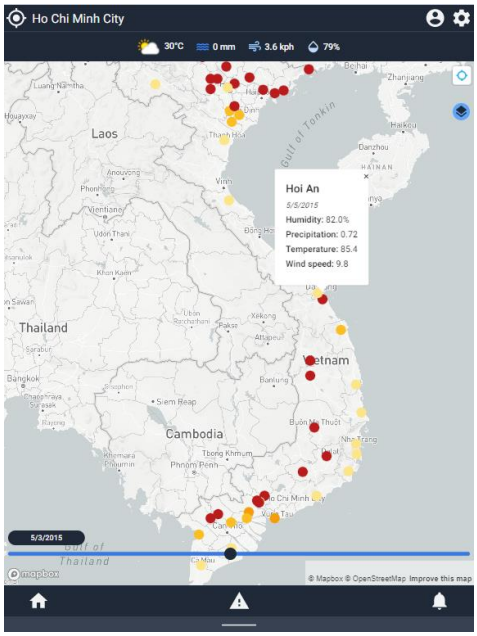


Fig. 5. Flood map visualization of city weather data

## 4 Discussion

### 4.1. Challenges

A major challenge with the development of FloodWatch was implementing scalable solutions both for the sake of the app itself and the developer experience. Developing scalable solutions for the app required an efficient means of querying data. For example, determining how to query daily city weather data by date efficiently was essential due to the large size of this data. Daily weather information from sixty cities

over the course of several months make up this data. This challenge was overcome using a feature of DynamoDB called global secondary indexes. This feature allows users to perform queries with a variety of attributes efficiently [17]. FloodWatch uses global secondary indexes to quickly query the city weather data by date when the user adjusts the range slider.

Another challenge was determining how to query only nearby flood report data. Initially, FloodWatch would query all this data and display it on the map. This would clearly not scale for larger amounts of data across the world. Instead, we developed an API endpoint which can query data within a certain radius (km) of a specific location. The latitude, longitude, and radius are input into the endpoint as query parameters. The endpoint uses the geopy Python library to calculate the geodesic distance two points [18].

As the project progressed and became more complex, development and testing became increasingly cumbersome. This was primarily due to the amount of data involved. This challenge became overcome by developing tools for uploading new data. Using boto3, an AWS software development kit for Python [19], I developed command line functions to automate the process of uploading new data. These functions allowed us to quickly upload mock flood report data for testing. They also enabled us to automatically upload large amounts of city weather data and view summary statistics of the existing data.

### 4.2. Connection to Coursework

Concepts from Advanced Software Development (CS 3240) have been essential to the development of FloodWatch. This course teaches the fundamentals of modern software development in an industry setting. Some of the concepts most relevant to this project from this course was requirements gathering, project management, architecture design, and modularity. These concepts guided the development of FloodWatch from the very initial planning stages to implementation.

Database Systems (CS 4750) helped inform our decisions during backend development. This course teaches concepts of database design and implementation. The concepts most relevant from this course were the optimization of database queries and NoSQL databases. These concepts allowed us to design with scalability to mind and informed our decision to use DynamoDB, a NoSQL database.

### 4.3. Future Work

Much of the foundation for FloodWatch has been built; however, there are several core features that have yet to be implemented. These features include authorization and authentication, image recognition, support for all locations, and flood forecasting. In order for the app to be usable by many people, there must be authentication to associate flood reports to the user who uploaded them. This can enable other features to be implemented such as a "like" button for flood reports or a "karma" score for evaluating a user's credibility and contributions. Naturally, authorization becomes necessary to limit a user's access to private information and unvetted flood reports.

Image recognition is needed to moderate newly created flood reports. These reports must be vetted before being uploaded as they may contain vulgar language or an inappropriate image. An automated image recognition pipeline would eliminate the need for a human to manually approve every new flood report. However, an interface for manually moderating flood reports is still necessary in the case that the image recognition model is unconfident. We plan to implement the image recognition pipeline using AWS Rekognition, a service for automating image and video analysis using machine learning [20].

It is essential that FloodWatch supports any location in order to maintain competitive advantage over existing apps. A big goal of FloodWatch's mission is to serve as a solution for anywhere in the world. Currently, the app only supports HCMC. However, expanding it to any location in the world would be relatively straightforward. Ideally, the app should be able to discern the location based on the user's current location. The user should also be able to manually set the location on the app.

Among all future improvements, flood forecasting is arguably the most important. A machine learning model for predicting flood severity on a given day is currently in progress but not yet integrated into the app. Integrating this model would involve developing an API which inputs a given date and location into the model and returns the predicted flood severity. On the frontend, these flood severity predictions would be visualized as a heatmap layer where a color gradient is used to indicate the severity of a region.

FloodWatch still has some way to go before being a robust, fully functional web application. The previously mentioned features may only be a few of the many needed to effectively accomplish the goals of FloodWatch. As its development progresses, new ideas and challenges will inevitably arise. However, the foundation we have built for FloodWatch will facilitate future developments.

## 5 Links

FloodWatch demo: https://floodwatch.netlify.app/

Project repository: https://github.com/floodwatch-org

## 6 Acknowledgements

This project has been funded by grants won by Professor Nguyen, Department of Computer Science. He has helped with defining the vision of the project, gaining support from our collaborators at HCMUS, and overseeing the project's progress.

# 7 REFERENCES


[1] C40 Cities Climate Leadership Group. (2018, February). *Sea level rise and coastal flooding*. C40 Cities. Retrieved May 15, 2022, from https://www.c40.org/what-we-do/scaling-up-climate-action/adaptation-water/the-future-we-dont-want/sea-level-rise/

[2] Ghose, T. (2013, August 18). *The 20 cities most vulnerable to flooding*. LiveScience. Retrieved May 15, 2022, from https://www.livescience.com/38956-most-vulnerable-cities-to-flooding.html

[3] Quynh, L. (2021, March 25). *Noose or life ring: Ho Chi Minh City's proposed Super Dyke*. Mekong Eye. Retrieved May 16, 2022, from https://www.mekongeye.com/2018/11/27/noose-or-life-ring-ho-chi-minh-citys-proposed-super-dyke/

[4] Thu, V. T. H., Tabata, T., Hiramatsu, K., Ngoc, T. A., & Harada, M. (2020). Assessing Impacts of Sea Level Rise and Sea Dike Construction on Salinity Regime in Can Gio Bay, South Vietnam. Journal of Waterway, Port, Coastal, and Ocean Engineering, 146(6), 05020006. doi:10.1061/(ASCE)WW.1943-5460.0000608

[5] Vna. (2019, April 19). *Hanoi to apply HSDC Maps in flood warning: Environment: Vietnam+ (vietnamplus)*. VietnamPlus. Retrieved May 15, 2022, from https://en.vietnamplus.vn/hanoi-to-apply-hsdc-maps-in-flood-warning/151326.vnp

[6] Ng, J. (2020, September 14). *Udi Maps is a Saigon weather app that helps drivers avoid flood-prone areas this rainy season*. TheSmartLocal Vietnam - Travel, Lifestyle, Culture & Language Guide. Retrieved May 15, 2022, from https://thesmartlocal.com/vietnam/udi-maps-weather-app/

[7] *About: Risk Factor*. Risk Factor. (n.d.). Retrieved May 15, 2022, from https://floodfactor.com/about

[8] *React – a JavaScript library for building user interfaces*. – A JavaScript library for building user interfaces. (n.d.). Retrieved May 13, 2022, from https://reactjs.org/

[9] *Rapidly build modern websites without ever leaving your HTML.* Tailwind CSS. (n.d.). Retrieved May 14, 2022, from https://tailwindcss.com/

[10] Mapbox GLJS. (n.d.). Retrieved May 14, 2022, from https://www.mapbox.com/mapbox-gljs

[11] *Documentation - AWS chalice*. Chalice. (n.d.). Retrieved May 15, 2022, from https://aws.github.io/chalice/index.html

[12] *What is Amazon DynamoDB?* Amazon Web Services. (n.d.). Retrieved May 14, 2022, from https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html

[13] *Develop & deploy the best web experiences in record time*. Netlify. (n.d.). Retrieved May 15, 2022, from https://www.netlify.com/

[14] *About Figma, the collaborative interface design tool.* Figma. (n.d.). Retrieved May 14, 2022, from https://www.figma.com/about/

[15] *Diagrams for confluence and jira*. draw.io. (2022, March 10). Retrieved May 14, 2022, from https://drawio-app.com/

[16] *Introduction to progressive web apps - progressive web apps (pwas): MDN*. Progressive web apps (PWAs) | MDN. (n.d.). Retrieved May 15, 2022, from https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Introduction

[17] *Using global secondary indexes in DynamoDB.* Amazon Web Services. (n.d.). Retrieved May 15, 2022, from https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/GSI.html

[18] *Welcome to GeoPy's documentation!* GeoPy. (n.d.). Retrieved May 15, 2022, from https://geopy.readthedocs.io/en/stable/#module-geopy.distance

[19] *AWS SDK for Python - Amazon Web Services (AWS)*. Amazon Web Services. (n.d.). Retrieved May 15, 2022, from https://aws.amazon.com/sdk-for-python/

[20] *Machine learning in the AWS cloud: Add intelligence to applications with Amazon Sagemaker and Amazon Rekognition*. Amazon Web Services. (2019). Retrieved May 15, 2022, from https://aws.amazon.com/rekognition/