Project Documentation: Flood Help

Possible Team

Introduction

“Flood Help” is a mobile application designed for use during flooding events in the greater Brisbane region. Flood Help provides users the ability to view a real-time picture of potential hazards during flooding events and the status of their loved ones.

The 2022 floods in Brisbane had devastating effects on the community with some 20,000 homes flooded, numerous public services not available, and 13 people losing their lives as a result of the flooding. In a critical situation such as this, when safety is at stake, real time and reliable information is key. People must be able to make informed choices about safe routes to take, when to stay home, or even when it is time to evacuate. This need for information revealed a glaring problem with the current flood information systems in Brisbane, namely that there are almost no sources where users can go to find specific and fine grained information. Our user surveys revealed that residents struggled to obtain accurate and timely reports of hazards such as flooded roads or rising flood waters, often relying on questionable social media posts to attempt to find the information they need. Reports from services such as BOM were too broad or simply not available, and reliable information was scattered.

Flood Help resolves this issue by providing an easy way for Brisbane residents to access a robust and centralised source of reliable information regarding ongoing or possible floods. Our app distinguishes itself from other current options by centralising information from official channels including Brisbane City Council and other reliable third party data sources, together with on the ground user reports where users can post photos and information about specific flood related hazards which others could use to take the necessary precautions. This collection of information means that users can see information about specific flooding events in their immediate vicinity such as flooded roads, or rivers breaking their banks, but also gain an overview of the situation as a whole through weather alerts from BOM or the map view of the city showing all reports. All together this serves to provide users a clear idea of the current flood related risks and hazards in their region.

Flood Help also provides a number of additional features that our user surveys revealed to be desirable, including the ability to track the location of specific users (primarily intended to be family) and the ability to see their current status which alleviates some of the stresses of dangerous situations, and access to the historical flood risk in different areas of Brisbane, allowing users to make informed choices during flooding events

Some of the effects of the 2022 Brisbane floods (Source: Validum)

Prototype Implementation

One of the core features of the application is the ability for users to create reports of hazards. These reports are location specific and are intended to reflect a specific occurrence such as a flooded road, river, fallen powerline, etc. Users provide some textual information about the hazard as well as an image of it. The importance of this feature is that it allows a fine grained view of flood related hazards. While a flood warning from the BOM might give people a general heads up about flooding, it does not provide user specific help. The hazard reporting system of Flood Help provides users the ability to make choices based on their own current situation and make choices based on what is relevant to them. For example, to know when a river has broken its banks and it is time to evacuate, or which roads are currently flooded and how they can get to safety, where exactly it is currently safe. Due to the chaotic nature of events such as floods, getting a clear picture of this information is only possible by centralising as much information as possible.

Official government issued flood alerts are also shown to all users. When an agency such as BOM issues a flood or severe weather warning this is sent to all users in the area. We decided to show these alerts as they are a reliable indicator of upcoming danger, and presenting them as a notification ensures that users who might not usually access such sources easily become aware of it. By showing these alerts on Flood App, we're providing an easier way for users to access official sources of data, and this prevents them from resorting to unreliable sources on social media platforms.

The main screen of the mobile application shows the user a map of Brisbane with icons for all reported hazards and the locations of the users which the current user has permission to track. This provides a quick overview of the current situation. The map is automatically updated at periodic intervals so that the user always has an up to date view.

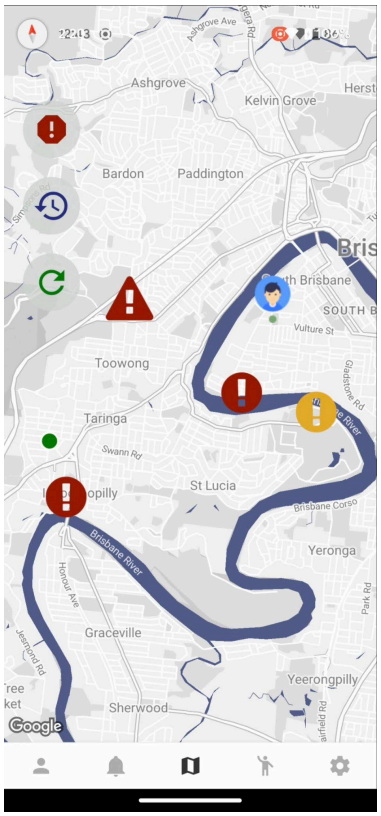
Flood Help places an emphasis on presenting users reliable information as this is critical for safety in dangerous situations such as floods. Users are provided a trustworthiness rating for each report which is calculated by corroborating them with other reports made nearby. If a large number of similar reports are made close to a report, then the report trustworthiness is increased. The use of images for reports additionally serves to help users gain a clear picture of the hazard. Information from official channels is also available, with the app providing real time weather warnings issued by the BOM, real time measurements of the height of various rivers, lakes and waterways in the region, and access to a map showing the historical risk of flooding in each area.

Another feature which makes Flood Help stand out is that it connects family members and friends, giving them the ability to see each other's location and track their safety status which is periodically updated. These connections are created by requesting a connection with another user via their email address, which allows easily finding the right user. Users can also request another user to update their status. Our user interviews reveal that residents found it difficult to stay connected. In the chaos, people often missed phone calls from loved ones, or were not able to contact them. Flood Help provides real-time location tracking and status updates to address this.

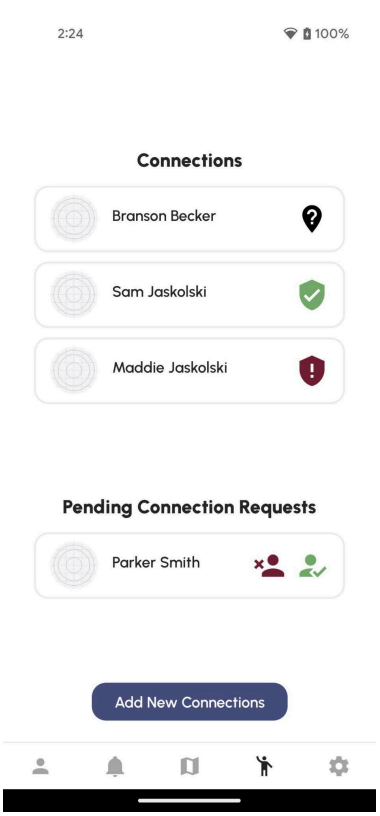
Users also have the option to look at flood likelihoods around Brisbane based on historical data. This application is supposed to give users all the information they can possibly have to support their

decision making, otherwise they could find themselves in a dangerous situation. Therefore, we implemented the additional option of looking at historical data to further inform users before they take a possibly life threatening decision.

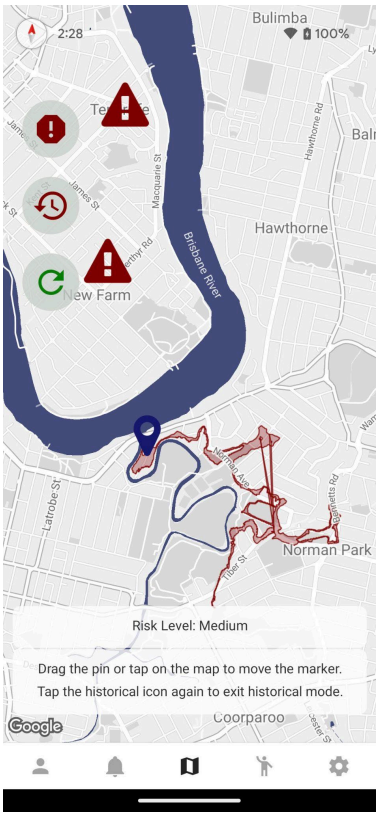
Users' locations are periodically updated so that their relationships can see their locations. If a large number of hazards is reported in a user's current location, they are sent a notification warning them of this. This serves to draw attention to potential danger, as they might not always check the application without prompting.



The main map of the application showing official warnings, user reports, and tracked connections



The connections screen showing users whose location is shown



A view of the historical flood risk in a specific region

The main apps that are used for flood tracking currently are weather apps like WeatherZone and the BOM app however these lack all the user reporting, connection features and don't have integrated flood history information. There are apps like flood alert watcher which contain the official warning locators on the map and historical flood data but also lack the user reporting on connection features. This additional community engagement feature on top of just showing data is what makes our app stand out from the rest. It allows the community to work together in dangerous situations instead of just relying on official data as this can have delays and not have the same level of fine grain hazard identifying that an entire community can achieve.

Tools & Technology

Front End

The front end of our mobile application provides a highly interactive and user-centric interface that enables users to manage flood-related activities, monitor hazards, receive notifications, track their connections, and manage profiles efficiently. Each of these features is powered by various tools and technologies, ensuring smooth functionality, data integrity, and real-time updates through a well-connected system with the backend. Here's a breakdown of the technologies and tools used, and why they were chosen:

Mobile Framework: React Native with Expo

React Native serves as the backbone of the mobile application, offering a cross-platform solution that allows us to develop for both Android and iOS with a single JavaScript codebase. While the project primarily targets Android users, React Native ensures that the application provides a native-like performance and user experience, while maintaining the flexibility to expand to iOS in the future. This choice was critical in reducing development time, as it allowed the team to focus on feature implementation rather than platform-specific customization. (see https://reactnative.dev/docs/getting-started for more information)

Expo was integrated to simplify development, testing, and deployment. Expo provides built-in tools and libraries, which streamline access to native features such as location services, push notifications, and camera functionality. This eliminates the need for extensive native code and reduces the complexity of managing platform-specific dependencies. Moreover, Expo facilitates testing on real devices by allowing easy builds and live testing, enhancing the development cycle's efficiency and ensuring that updates to the application can be quickly tested in real-world environments.(see https://docs.expo.dev/get-started/introduction/ for more information)

Google Maps API

The Google Maps API is at the core of the application's mapping functionalities, which include real-time flood hazard reports, official flood alerts, and historical flood data visualisation. We integrated this API with React Native using the react-native-maps package, which provides a seamless interface for displaying interactive maps, markers, and polygons.

The choice of Google Maps is driven by its reliability, global reach, and comprehensive toolset. It provides features such as geocoding (to convert addresses into coordinates), reverse geocoding (to convert coordinates back into addresses), and route mapping, ensuring accurate geographical data is presented to users. In addition to showing real-time locations of flood zones, the maps also display user connections, allowing users to view their friends' locations, receive check-ins, and view their status during critical events.

Google Maps also supports custom markers and overlays, allowing us to differentiate between hazard reports, official alerts, and user-generated check-ins, enhancing the overall user experience and interaction with the map interface.

Backend Communication

The front end communicates extensively with the backend via RESTful APIs to handle user interactions, such as submitting hazard reports, fetching flood alerts, managing connections, and updating profiles. Each action initiated by the user sends an HTTP request to the backend, where data processing occurs, and the relevant information is returned as JSON.

We implemented these communications using JavaScript's fetch API within React Native. This allows the application to send and receive data asynchronously, ensuring real-time updates and synchronisation with the backend system. Whether it's updating user locations, submitting new hazard reports, or retrieving flood alerts, the use of RESTful API standards ensures a seamless data flow between the front end and back end.

State Management: React Hooks & Context API

For managing application state, we employed React's Hooks and Context API. These tools are essential in ensuring the application maintains its responsiveness and data flow, particularly when handling user authentication, real-time data fetching, and updates.

- **React Hooks**: The use of useState and useEffect is fundamental in managing the lifecycle of components. useState stores critical information like flood reports, user location, and connection status, while useEffect is used to handle side effects such as fetching real-time data from the backend or triggering specific actions when the app's state changes. This allows the application to remain responsive and dynamically update based on user interactions or new data from the backend.
- **Context API**: The Context API is used to maintain global state across the application, such as user authentication and session management. This ensures that user-specific data (e.g., profile status, connections, and past reports) is consistently available across different components, without the need for prop drilling. Additionally, the Context API enables smoother management of user sessions, ensuring seamless navigation and state retention as users move between different sections of the app.

UI/UX Design: Styled Components

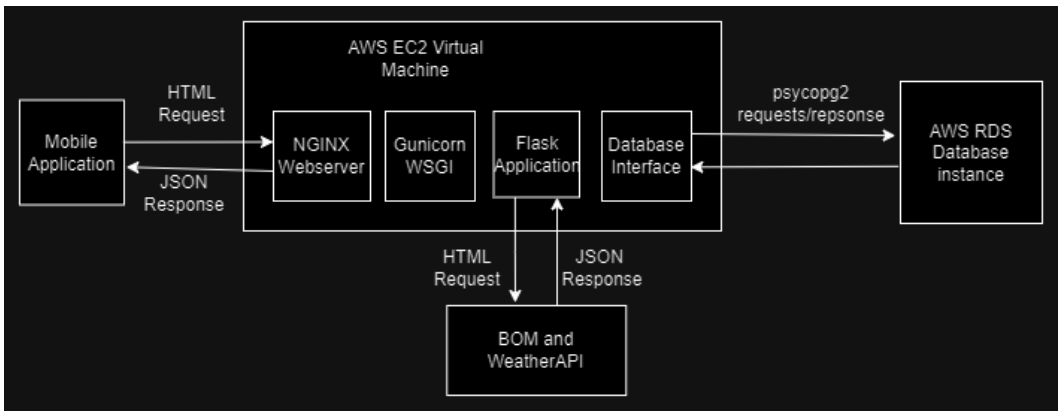
We adopted Styled Components to manage the styling of the application. Styled Components allow us to write CSS-in-JS, meaning we can define component-specific styles directly within the JavaScript code that handles the component's logic. This approach offers several advantages:

- **Component-Level Styles**: Each component in the app can have its own encapsulated styles, which helps maintain modularity and avoid style conflicts. This is crucial as the app grows in complexity and new features are added. Encapsulated styling ensures that changes in one component's design don't inadvertently affect others.
- **Dynamic Theming**: Styled Components make it easy to implement dynamic theming, such as switching between light and dark themes. This feature improves user experience, especially

when navigating the app in different environments (e.g., bright sunlight vs. night-time usage). Themes are applied globally, but the design remains responsive and consistent across all screens and components, enhancing both aesthetic appeal and usability.

Back End

The back end of the application handles information storage, retrieval, and application logic. When a user does something on the front end (mobile application) that affects the state of the database (e.g. create a hazard report), the mobile application makes a HTTP request to the backend API, following the REST API standards. This is received by the backend and passed through a series of modules (discussed in next sections), evaluated and acted upon. If required, a response in JSON format is then generated and returned to the front end application. A basic diagram of the information flow of the whole application is shown below.



API Server

Flask is an open source python api server library. It was chosen as the core of the api server due to it being extremely intuitive and simple, made in python which we all already knew, had a large user base so it was easy to find answers to common questions and good documentation. Another key benefit of flask is that it's extremely lightweight and minimal since our need for the api server is fairly minimal just needing to be able to set up routes and send and receive JSON information. This allowed us to quickly learn and set up our first working route within the first day of starting the project since the setup was 4 lines of code and then just defining the route. By choosing flask we were able to jump straight into actually implementing the features we wanted. Due to simplicity again when it came to start deploying our api in more of a production environment all we had to do was disable debug mode and run it through Gunicorn and it worked flawlessly. However this simplicity does come with some caveats, one of the major ones being performance. Since it's created in python it doesn't have the same performance capabilities of the Java libraries however due to the nature of our implementation where all crucial data is stored in a separate database we could easily scale up the number of instances of the api we were running with a few small tweaks. By scaling up this would alleviate the performance limitations allowing us to support a growing user base. Can read more about flask API here (https://flask.palletsprojects.com/en/3.0.x/).
The Flask server was hosted on an Amazon EC2 instance. It was chosen to go with a cloud implementation over self hosting because it means that it is always available and it removed one extra

point of failure from the application as AWS almost never goes down. It also has the benefit of easy scaling up, we can easily change the resources available on the node or create new nodes as the requirements or usage of the api increase. AWS was chosen in particular due to its pricing and flexibility. Amazon's AWS EC2 was cheapest out of the 3 major providers google clouds compute engine and microsoft azure vms, and since its needs to always needs to be running and doesn't need any other outside connections or features other than the database there was no reason to choose any of the others. For example one of the major benefits of GCP for development is that you only get charge for the time the vm is running compared to aws charging you for the entire the vm exists however due to out application being an API that needs to be running at all the times for the app to be able to connect on demand this benefit was redundant.

While Flask does have out of the box capability to receive web requests, it is not recommended to use this in production environments as there is no security, load balancing, multithreading, or static content serving. We chose to use Nginx and Gunicorn to handle these features. Gunicorn is a Python WSGI (Web Server Gateway Interface) HTTP server that serves the Python application. It's lightweight, easy to configure, and can handle concurrent requests through its multi-threading and asynchronous worker model. Additionally, it is required as an interface between the Flask application and Nginx (see https://docs.gunicorn.org/en/stable/ for more information). Nginx is a high-performance, reverse proxy server that can handle large numbers of simultaneous connections, efficiently serve static content (such as images), and provide additional security features such as HTTPS connections. It also acts as a load balancer and can forward incoming traffic to Gunicorn. By placing Nginx in front of Gunicorn, Nginx handles the client connections, reduces the load on Gunicorn, and protects it from certain attacks (e.g., DDoS). See (https://nginx.org/en/docs/) for more information. Together these serve to increase the robustness, efficiency, and security of the backend. Currently they are configured in a basic setup, but can easily be altered to provide almost limitless scaling.

Database

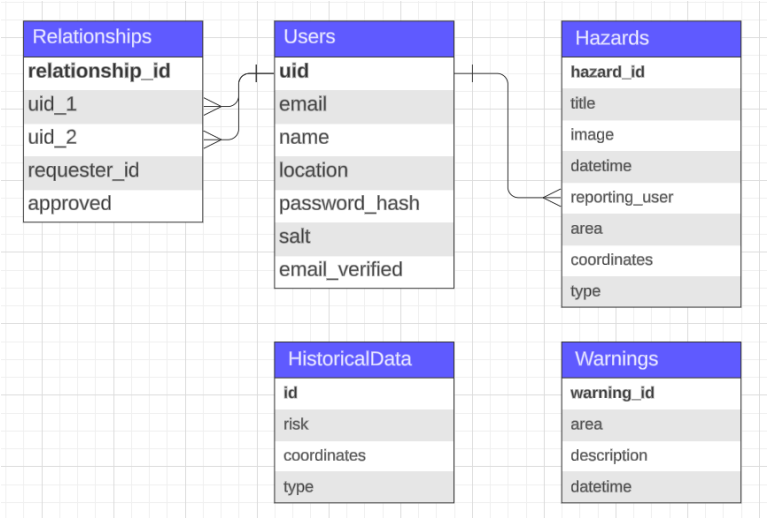
We elected to use Postgres as our database management system (DBMS) as is it is free and open source, has a large ecosystem of external libraries (such as psycopg2 which was used in the interface), and has excellent official documentation as well as a large amount of troubleshooting information available due to its widespread use. Postgres is a relational database engine, which allows the storage of structured data which needs to be associated with other pieces of data (such as associating a report with a specific user). Postgres also offers multiple advantages over other relational DBMS's such as complex data types, robust security features, and a high degree of customisation and configuration, as well as numerous other aspects (see https://www.prisma.io/dataguide/postgresql/benefits-of-postgresql for an overview).

Postgres is more scalable than other comparable options such as MySQL as it allows additional configuration options. Since our current target audience is limited to the greater Brisbane area (i.e. a few million users at most), the current configuration will be more than sufficient. Should we decide to scale the application further to potentially multiple countries and an extraordinarily large number of users, the database may run into problems with scalability and a different approach such as using a distributed database management system, or a NoSQL based approach may be advisable, however multiple changes in rest of the backend will also be required.

We elected to host the database on a dedicated instance of AWS Relational Database System (RDS) as this helps take the load of the EC2 virtual machine. As well as hosting the actual database, RDS provides features such as monitoring, access control, automatic backups, and automatic scaling (see https://aws.amazon.com/ec2/features/). Since we were already using AWS EC2 to host the API, using another Amazon service allows extremely easy integration between the two. The RDS instance was configured to only allow access from the within the Amazon Virtual Private Cloud in which both the EC2 instance and the RDS instance were deployed (see https://docs.aws.amazon.com/vpc/latest/userguide/what-is-amazon-vpc.html for more information). The effect of this is that the Database is only accessible from EC2 which assists with security. Using a dedicated instance of RDS to host the database, rather than hosting in on EC2, provides the advantages of improved performance as the load is distributed, as well as easy scalability due to AWS's serverless architecture. It is possible to increase the storage space, CPU, RAM and almost every other feature at the click of a button in the AWS console. A guide on how a database instance can be set up can be found at https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_CreateDBInstance.html.

The interface between the Flask API application and the database was implemented as a python class which uses the psycopg2 library to connect to and query the database. The class establishes a remote connection, and allows the use of a specific set of methods created by us to update, retrieve, insert, and delete the data which is required by the application. These methods query the database and use prepared statements which ensure that any user input which is passed to the database is treated as an escaped string, thereby eliminating the chance of SQL Injection attacks (see https://portswigger.net/web-security/sql-injection). Additionally, this adds a layer of abstraction and ensures that database operations are correctly executed, reducing the risk of a developer making a mistake. For more information on the psycopg2 library see (https://www.psycopg.org/docs/).

The schema of the database was kept fairly simple, however a relational database allows the creation of highly complex and structured data sets, allowing the easy adaptation of the project beyond a prototype. The data stored in the database is user information (Users), relationships between users who can see each others locations and statuses (Relationships), user reported hazards (Hazards), warnings issued by the BOM (Warnings), and the historical flood risk of various areas in the Brisbane region (HistoricalData). The exact schema definition of the database, including types, triggers, and constraints can be found at floodhelp/BackEnd/Database/schema.sql. The general structure is as follows:



The structure of the Flood Help database

External Data

For external data, we first used historical datasets supplied by the Brisbane City Council (see https://data.brisbane.qld.gov.au/explore/dataset/flood-awareness-flood-risk-overall/information/). We wanted historical data, which would act as secondary information to support the real-time data, because users should be aware of the risk which is present in certain areas and need all the information they can get before making any important decisions that could potentially save their lives. We decided to use Brisbane City Council to get this data as it is an official source directly from the government and hence highly reputable. After extracting the datasets, we filtered them by only taking the areas with a significant flood risk, which we considered to be either “Medium” or “High”. We then inserted the resulting data into the database and added routes to allow the front end application to access it.

We also made use of real-time data since it would provide the users with crucial live information in the case of flash flooding or the lack of user reports. The first type of real-time data we used was river height classifications from the Brisbane City Council (see https://data.brisbane.qld.gov.au/explore/dataset/telemetry-sensors-rainfall-and-stream-heights/information/). This was chosen based on historical evidence of Brisbane floods being situated near its rivers, and the data was taken from the Brisbane City Council for the reasons discussed with the historical data. The data itself consisted of information, which was updated every five minutes, regarding forty stations around Brisbane that measure river heights. The API in the Brisbane City Council website was clear and accessible, and calls were made to it every hour to fetch the updated datasets. The information was then filtered to only take the latest row of station coordinates and river height values. To make the flood classifications, we used a document, taken from the Bureau of Meteorology website, that has a flood classification for each corresponding height interval for all stations in Brisbane. It is worth noting that this has been implemented in the back end but not on the front end, so although it isn’t shown on the application itself, it is ready to be implemented for future development.

The second type of real-time data we used was official alerts from WeatherAPI (see https://www.weatherapi.com). These are government-issued alerts. As such, they have a top priority when it comes to all information made available to the user. Besides its high credibility, we decided to include this because one of the main reasons Flood Help was made is so that users wouldn’t have to

search on social media for alerts like these. We used WeatherAPI because it's a commonly used website to extract this kind of information, thus making it more trustworthy. We would have extracted the alerts from government websites, but those that do issue these alerts (like BoM) don't have an API to retrieve them. The alerts were extracted and then updated in the database every five minutes.

Project Management & Collaboration Process

As a team, we assigned different roles to each other based on our respective skill sets. Like many professional environments, we decided to go with a top-down approach when it came to splitting work between members. The idea is that it increases efficiency by maximising organisation. Firstly, the team was split evenly into members who worked on the back end and members who worked on the front end. Furthermore, the roles were split even within those sub-team; for example, in the backend, one member worked on the API, another on the database, and another on extracting data from external sources. The front end, however, was split based on the different screens shown in the application; one member was in charge of developing the map and reporting screens, while another was responsible for screens such as login and profile.

We created the milestones based on the course assessments. The first major assessment was a presentation, so we made sure that members working on the front end had a prototype ready by then to show how basic features would work. For the backend, smaller milestones were set in place such as setting up the database, connecting it with the api, etc…

For communication, we used Microsoft Teams. We met every Friday morning and occasionally through video calls on Wednesday nights. For efficiency, our meeting only included relevant topics that were prioritised based on the week. In addition, we used shared google documents to collectively collaborate on the different reports throughout. By doing this, we were able to see each other's changes in real time. We applied this method of thinking to the codebase as well. Github allowed us to effectively add our individual contributions without facing many issues. To comply with standard professional programming practices, we also regularly committed our work and respectively included detailed descriptions. This gave our work transparency, which allowed members to better understand each other's contributions, thus increasing code quality in the long term and adding a crucial level of trust in the team. All methods included descriptions to increase readability of the code, and a README was written to ease the communication between the team itself and whoever wants to review the codebase.

The management methodology we adopted was agile, this was used in conjunction with out miles To make sure we were achieving our goals. It involved breaking down what needed to be achieved before each milestone into weekly sprints (goals). The first stage of each sprint was deciding on the requirements, this was done during the end of the Friday studio sessions where we would discuss what the goal was for the next week and what were the requirements that needed to be met to achieve that goal. The requirements were then split between each member of the group to complete the next stages: design, develop and test. These were mostly done individually unless the design had wide reaching effects in which this was discussed online to ensure their design fit in with what others were

doing. Once they were finished they deployed their solution through a pull request on github. Each pull request was reviewed by another person on their same team (backend/frontend) to ensure it was meeting the requirements. At the start of each studio session we would look at the changes made by everyone that week if there were any issues or delays, and the weekly plan was adjusted to ensure we were still meeting our overall goal.

In a group, we would reflect on feedback given to us by tutors, and then we would discuss the possible ways to improve.

Ethics, Security & Data Privacy

Incidents where user data leaks occur have inspired us to take specific actions against these kinds of attacks (see https://www.bbc.com/news/business-41493494 for an example of such an incident). Since users must initially sign up after downloading the app, then having to share personal information is unavoidable; therefore, protecting that information and making sure we only ask for what's necessary to ensure data privacy is of utmost importance to us. Highlighting the steps we took to achieve this, all user information stored is encrypted, users are only required to provide their email address, and user locations aren't stored anywhere in the database. Failure to take these steps would lead to a breach of trust between us and the users, in addition to putting them at the risk of phishing attacks due to leaked emails or at the risk of more serious offences due to leaked location data.

One ethical risk we've been actively mitigating is the possibility of flood risks being wrong or inaccurate. For example, this could occur if the shown user reports in the app are outdated. Another example is the possibility of the alerts taken from external sources being unreliable in terms of how factual they are or how recent they are. We've solved these issues by keeping track of how long the user reports have been posted and by only extracting official flood alerts from known sources.

Another ethical risk arises from the use of data from external sources; this specifically has to do with copyright and complying with terms of service. All sources we're extracting data from require the use to be for non-commercial purposes. In addition, some sources, like the Bureau of Meteorology, require that a link to the original source be present when the data is shown in Flood Help.

The security of our application is paramount, because Flood Help could indeed be the difference between life and death for many Brisbane residents, therefore we cannot afford to have the functionality of our application be disrupted at any point in time. This could potentially happen if external actors decided to spam false user reports or make any attempt at slowing down our server.

Possible Risk	**Type**	**Likelihood**	**Severity of Impact**	**Minimisation strategy**
Personal Information Leak -> user data such as names email address are leaked to the public through (database/api breach)	Data privacy	Medium	High	Only require minimal information for sign up. Other information like "location" will not be kept on servers. Location just flows through the server and is stored on devices.
External actors wanting to harm the app by spamming the "user data" with false information (this can be done at large using bots)	Security	Low	Medium	Require identification using their email and requires user logins to access any of the content..

Team member/s dropping out for any particular reason -> team member/s having to leave for medical reasons (sickness,death), dropping out (laziness, conflict) or Forced to relocate, etc…	Project	Low	Medium	Group members have been chosen in such a way that our skill sets are different enough so that we can all contribute, but also with enough similarity such that if a team member drops out, others can help take over. In addition, all group members are comfortable with adapting to new skills and learning things outside of their main skill sets.
Basing flood risk results on wrong or outdated information, thus potentially causing unnecessary stress for app users and possibly endangering them.	Ethical	Low	High	Utilize data from well known and reliable external sources (Weather API, using BOM data). In addition, using data from the users themselves (user reports)

Evaluation

To evaluate our app with stakeholders (people who could be involved and ideally has been involved in a flood in Brisbane) we followed a procedure to try and make the results as objective as possible. Originally the application was given to a user and told to try everything out in the app to see if they could complete the following steps without saying anything

1. Create account
2. Login to account
3. Find and get information of hazard on the map
4. Find historical data on map
5. Find validation score of hazard
6. Create a hazard
7. Create/Accept a relationship
8. Find other users location map
9. Find and get notifications
10. Check-In and view someones status

Whatever they did not complete they were instructed to do. If they couldn't figure out how to do it after being instructed they were helped through the process of doing so. Any pain points the user verbalised or anything that we observed them struggling with while using the app was recorded along with what users need to be instructed to do and what we had to help them to do. Additionally they were also asked about what they think could make the app better. Anything multiple users couldnt find was evaluated as not being prominent or clear enough and anything they struggled to do was seen as needing to be more intuitive.

Results from user testing:

User	**Completed self guided**	**Completed with promoting Promoted**	**Completed with Assisted**	**Pain Points**	**Future Recommendation**
Zoe	5/10	8/10 -having to go back to login after creating account	10/10 -understood everything after small assistance -need help with accepting a connection since they didn't refresh it -need help to find historical	When accepting someone's connection request it does not auto refresh so it wasn't showing the connection was accepted even	-instagram like notes system for status so can enter your own status instead of the set ones -be able to change profile picture

			data on map because they were sure how the marker worked	though it was. Had to manually refresh page	
Jacob	8/10	9/10 -Had to bring up that the check-in status is under the user profile	10/10 -need help to access historical data because where they were placing the marker isn't showing anything	Tapping a hazard it took a while to show the info. Not auto refreshing connections.	Make it so you can select an area to define as a hazard instead of just a marker
Alex	6/10 -Name placeholder text in sign-in screen should be a name instead of Flood Help.	7/10 -Unsure why flood needs a title. Also Flood type should be on top.	7/10 -When pressing on safe/unsafe icons next to user in connection screen, interactions were expected, however no icon interaction was implemented	-The app is almost too streamlined	- While the app has a clean look, clarifications on each screen/icon should be added, a site tour, even for the first time users only, might be helpful
Jordan	9/10	9/10 -didn't help by prompting	10/10 -needed help finding status pushes, they were trying to find somewhere in the accounts or connection not notifications	- took a bit to find status update pushes	-have predicted flood levels based on the history data and current rain
Sam	8/10	9/10 -having to go back to login after creating account aswell	10/10 -didn't need any assistance to do everything	-struggle a bit to understand connections since a lot of impacts of adding connections happen outside of the connections area	- automatically inform emergency services if status is unsafe

Conclusion & Recommendations

During the Brisbane floods in 2022, people resorted to using social media and other similar sources for the purpose of making important decisions to avert danger; “Flood Help” solves this issue by providing the residents of Brisbane with an easier way of accessing reliable sources of information regarding flood hazards. The application allows users to send flood warnings to each other, in addition to showing them historical information and official government alerts to further support their decision making. We did this by first setting up an API server using flask and a database using postgres. React Native was used for the front end of the application. Furthermore, we gathered the external data from reliable sources such as the Brisbane City Council, BOM, and WeatherAPI. We also made sure to mitigate ethical, security, and data privacy risks, especially since the application requires the storage of users’ personal information such as emails.

For future development, markers should be replaced with areas, as that would improve clarification for other users. Adding future flood predictions and automatic emergency services contact could help users escape unsafe situations. In addition, tutorial-like features should be added so that users could have an easier time using the application. The application is designed with an Android phone in mind, so future development should also focus on expanding the scope by having it fully work on other devices such as the iPhone.

Due to the abundance of rivers in Brisbane, the main cause of floods is the overflowing of the rivers. As a result, communicating the flood status of these rivers to the users is vital. Stations around Brisbane show the stream height status of these rivers, and we have access to that data through the Brisbane City Council public datasets. This data is currently not integrated in the front end, but will be made available in future versions.

The primary points of the brief and statement of work were satisfied, and we have a clear roadmap for how to continue. The design was created with regards to the user experience, and the implementation with regards to efficiency, scalability and robustness. We conclude that the project is a successful prototype implementing the primary features of the project.

Bibliography

BBC (2017). Yahoo 2013 data breach hit ‘all accounts’. *BBC News*. [online] 3 Oct. Available at: https://www.bbc.com/news/business-41493494.

Bom (2019). *Australia’s official weather forecasts & weather radar - Bureau of Meteorology*. [online] Bom.gov.au. Available at: http://www.bom.gov.au.

Wikipedia contributors. (2023, October 12). *2022 eastern Australia floods.* Wikipedia. https://en.wikipedia.org/wiki/2022_eastern_Australia_floods

Benson, J. (2020). Flood Alert Watcher. [online] App Store. Available at: https://apps.apple.com/au/app/flood-alert-watcher/id1517218014 [Accessed 18 Oct. 2024]

Griffith University. (2023, May 23). *Why was Brisbane’s 2022 flood different?* Griffith News. https://news.griffith.edu.au/2023/05/23/why-was-brisbanes-2022-flood-different/

OneSignal. (2023, March 15). *Lessons learned from 5 years of scaling PostgreSQL.* OneSignal. https://onesignal.com/blog/lessons-learned-from-5-years-of-scaling-postgresql/

Quest Software. (n.d.). *What is PostgreSQL?* Quest. https://www.quest.com/learn/what-is-postgresql.aspx

Amazon Web Services. (n.d.). *The difference between MySQL vs PostgreSQL.* AWS. https://aws.amazon.com/compare/the-difference-between-mysql-vs-postgresql/

Amazon Web Services. (n.d.). *Amazon RDS (Relational Database Service).* AWS. https://aws.amazon.com/rds/

Brisbane City Council. (n.d.). *Flood awareness: Flood risk overall.* Brisbane Open Data. https://data.brisbane.qld.gov.au/explore/dataset/flood-awareness-flood-risk-overall/information/

Brisbane City Council. (n.d.). *Telemetry sensors: Rainfall and stream heights.* Brisbane Open Data. https://data.brisbane.qld.gov.au/explore/dataset/telemetry-sensors-rainfall-and-stream-heights/information/

WeatherAPI. (n.d.). *Weather API.* https://www.weatherapi.com

Weatherzone.com.au. (2015). *Weather - Australia 7 day forecasts and weather radar - Weatherzone*. [online] Available at: https://www.weatherzone.com.au.

Validum Institute. (2022, March). *2022 Queensland floods* [Image]. Validum Institute. https://www.validum.edu.au/wp-content/uploads/2022/03/2022-Queensland-floods.jpg

PortSwigger. (n.d.). *SQL injection*. PortSwigger Web Security.
https://portswigger.net/web-security/sql-injection

https://www.bbc.com/news/business-41493494

Appendix 1:

ChatGPT Prompt:
Create a consistent bibliography of the following resources
https://en.wikipedia.org/wiki/2022_eastern_Australia_floods#:~:text=In%20Brisbane%20177%20suburbs%20were,buses%2C%20and%20City%20Cat%20ferries.
https://news.griffith.edu.au/2023/05/23/why-was-brisbanes-2022-flood-different/
https://onesignal.com/blog/lessons-learned-from-5-years-of-scaling-postgresql/
https://www.quest.com/learn/what-is-postgresql.aspx#:~:text=Postgres%20is%20a%20popular%20choice,search%20and%20JSON%20data%20types.&text=Postgres%20can%20handle%20high%2Dtraffic,gaming%20and%20social%20media%20applications.
https://aws.amazon.com/compare/the-difference-between-mysql-vs-postgresql/
https://aws.amazon.com/rds/
https://data.brisbane.qld.gov.au/explore/dataset/flood-awareness-flood-risk-overall/information/
https://data.brisbane.qld.gov.au/explore/dataset/telemetry-sensors-rainfall-and-stream-heights/information/ https://www.weatherapi.com
https://www.validum.edu.au/wp-content/uploads/2022/03/2022-Queensland-floods.jpg

Response:
Here's a consistent bibliography for the resources you provided, formatted in APA style:

1. Wikipedia contributors. (2023, October 12). *2022 eastern Australia floods*. Wikipedia. https://en.wikipedia.org/wiki/2022_eastern_Australia_floods
2. Griffith University. (2023, May 23). *Why was Brisbane's 2022 flood different?* Griffith News. https://news.griffith.edu.au/2023/05/23/why-was-brisbanes-2022-flood-different/
3. OneSignal. (2023, March 15). *Lessons learned from 5 years of scaling PostgreSQL*. OneSignal. https://onesignal.com/blog/lessons-learned-from-5-years-of-scaling-postgresql/
4. Quest Software. (n.d.). *What is PostgreSQL?* Quest. https://www.quest.com/learn/what-is-postgresql.aspx
5. Amazon Web Services. (n.d.). *The difference between MySQL vs PostgreSQL*. AWS. https://aws.amazon.com/compare/the-difference-between-mysql-vs-postgresql/
6. Amazon Web Services. (n.d.). *Amazon RDS (Relational Database Service)*. AWS. https://aws.amazon.com/rds/
7. Brisbane City Council. (n.d.). *Flood awareness: Flood risk overall*. Brisbane Open Data. https://data.brisbane.qld.gov.au/explore/dataset/flood-awareness-flood-risk-overall/information/
8. Brisbane City Council. (n.d.). *Telemetry sensors: Rainfall and stream heights*. Brisbane Open Data. https://data.brisbane.qld.gov.au/explore/dataset/telemetry-sensors-rainfall-and-stream-heights/information/
9. WeatherAPI. (n.d.). *Weather API*. https://www.weatherapi.com
10. Validum Institute. (2022, March). *2022 Queensland floods* [Image]. Validum Institute. https://www.validum.edu.au/wp-content/uploads/2022/03/2022-Queensland-floods.jpg