SW Engineering CSC648-848-05 Spring 2025

Crisis Relief

Team #5

Team Members:

1. **Anshaj Vats** - Team Lead (avats@sfsu.edu)
2. **Francis Aviles** - Database Administrator
3. **Ayesha Irum** - Backend Lead
4. **Geoart Corral** - GitHub Master
5. **Karla Cardenas Andrade** - Frontend Lead
6. **Kyle Nguyen** - Technical Writer

Milestone 2

Milestone	Version	Date
Milestone 2	Version 2	April 17, 2025
Milestone 2	Version 1	March 30, 2025
Milestone 1	Version 2	March 20, 2025
Milestone 1	Version 1	February 20, 2025

Table of Contents

Data Definitions

1. Users

Attribute Name	Data Type	Description
user_id	UUID (Primary key)	Identifier for user
full_name	VARCHAR(50)	User's full name
email	VARCHAR(100)	Email used for registration and log in
password_hash	VARCHAR(100)	Hashed password for security
phone_num	VARCHAR(10)	User's phone number (optional?)
role	ENUM(User,Relief Worker, Admin)	Determines access level

User Roles and Privileges

Role	Can Search for Resources	Can Post Updates	Can Approve Updates	Can Approve Updates
User	Yes	No	No	No
Relief Worker	Yes	Yes	No	No
Admin	Yes	Yes	Yes	Yes

2. Emergency Resources

Attribute Name	Data Type	Description
resource_id	UUID (Primary Key)	Unique identifier for each resource
name	VARCHAR(50)	Name of the resource
type	ENUM(Shelter, FoodBank, MedicalCenter, MentalHealthSupport, Supply Station	Type of resource that it is
address	VARCHAR(100)	Street address of resource
city	VARCHAR(100)	City of resource
state	VARCHAR(25)	State of resource
zip_code	VARCHAR(10)	ZIP code of resource
latitude	DECIMAL(9,6)	Latitude of resource
longitude	DECIMAL(9,6)	Longitude of resource
capacity	INT	Total Capacity of the resource (For places like shelter)
available_slots	INT	Current availability
contact_number	VARCHAR(15)	Contact Number
website_url	VARCHAR(255)	Website or official information (if they have)
open_hours	VARCHAR(100)	Operating hours
status	ENUM(Open, Full, Closed, Unknown)	Indicates current status
last_updated_at	TIMESTAMP	Timestamp of the last update

3. Weather Alerts

Attribute Name	Data Type	Description
alert_id	UUID (Primary Key)	Unique identifier for each alert
region	VARCHAR(100)	Region affected by the weather alert
alert_type	ENUM(Storm, Hurricane, Flood, Heatwave, Wildfire, Tornado, Other)	Type of weather alert
severity	ENUM(Low, Moderate Severe, Extreme)	Severity of the weather event
description	TEXT	Description of the weather event
start_time	TIMESTAMP	When the weather event is expected to begin
end_time	TIMESTAMP	When the weather event is expected to end
created_at	TIMESTAMP	Timestamp when the alert was created

4. Notifications and Alerts

Attribute Name	Data Type	Description
notification_id	UUID (Primary Key)	Unique identifier for each notification
user_id	UUID (Foreign Key)	User receiving the notification
message	TEXT	Content of notification
type	ENUM(WeatherAlert, ResourceUpdate, SystemUpdate)	Category of notification
is_read	BOOLEAN	Indicates if the user has seen the notification
created_at	TIMESTAMP	Timestamp when the notification was sent

5. Feedback

Attribute Name	Data Type	Description
review_id	UUID (Primary Key)	Unique identifier for each review
user_id	UUID (Foreign Key)	User who wrote the review
resource_id	UUID (Foreign Key)	Resource being reviewed
rating	INT	Rating (1-5)
comment	TEXT	User feedback
created_at	TIMESTAMP	Date and time the review was submitted

6. Search and Personal Data

Attribute Name	Data Type	Description
search_id	UUID (Primary Key)	Unique identifier for each search query
user_id	UUID (Foreign Key)	User who performed the search
query	VARCHAR(150)	User’s search input
results_count	INT	Number of results returned
selected_result_id	UUID (Foreign Key)	ID of the resource selected by the user
timestamp	TIMESTAMP	Time of the search

Prioritized High Level Functional Requirements

Priority 1 – Critical (API Integrations & Core System Features)

1. API Integrations for Emergency Resources

- 1.1 The system shall integrate Google Maps API to provide real-time directions to emergency shelters, food banks, and medical centers.
- 1.2 The system shall integrate OpenWeatherMap API to display weather conditions that affect access to resources.
- 1.3 The system shall integrate real-time APIs from relief organizations (if available) to fetch live shelter and food bank data.
- 1.4 The system shall allow users to search for nearby emergency shelters based on real-time API data.
- 1.5 The system shall display real-time availability of emergency shelters retrieved from API updates.
- 1.6 The system shall notify users of real-time updates on shelter capacity, utilizing data from APIs or relief workers.
- 1.7 The system shall display real-time food bank inventory using APIs or relief worker updates.
- 1.8 The system shall notify users of weather alerts that impact food bank accessibility, using OpenWeatherMap API.
- 1.9 The system shall allow users to search for medical resources and provide real-time availability using API data.
- 1.10 The system shall provide directions to the nearest emergency resources using Google Maps API.
- 1.11 The system shall display real-time traffic conditions affecting travel time to emergency resources.

2. User Authentication

- 2.1 The system shall allow users to register using their email and password.
- 2.2 The system shall allow users to log in securely using their registered credentials.
- 2.3 The system shall implement session timeout functionality to ensure secure access.
- 2.4 Passwords shall be encrypted securely using industry-standard hashing (e.g., bcrypt).
- 2.5 The system shall ensure that user sessions are managed securely using JWT tokens.

3. Notifications & Alerts (API-Based)

- 3.1 The system shall alert users with real-time weather updates using OpenWeatherMap API.
- 3.2 The system shall notify users of real-time updates on shelter capacity, based on API or relief worker input.
- 3.3 The system shall notify users of real-time updates on food bank availability and restocking, using API integrations.
- 3.4 The system shall notify users of medical resource availability changes using API data.
- 3.5 The system shall allow users to set alerts for weather conditions impacting resource access.

Priority 2 – Important (User Input & Reviews)

4. User-Generated Data & Feedback

- 4.1 The system shall allow relief workers to update shelter capacity manually if API data is unavailable.
- 4.2 The system shall allow relief workers to update food bank inventory manually.
- 4.3 The system shall allow relief workers to mark shortages of supplies (e.g., water, blankets).
- 4.4 The system shall allow relief workers to update medical center availability.

5. User Experience & Review System

- 5.1 The system shall allow users to submit reviews and ratings for shelters, food banks, and medical centers.
- 5.2 The system shall display average ratings and user reviews for each resource.
- 5.3 The system shall allow users to flag inaccurate resource information for admin review.
- 5.4 The system shall allow users to customize notification preferences (e.g., food bank restocking alerts, weather changes).

6. Mobile Optimization

- 6.1 The system shall provide a responsive mobile-friendly design.
- 6.2 The system shall allow users to save favorite locations for quick access.
- 6.3 The system shall allow users to receive notifications on new shelters or food banks in their area.
- 6.4 The system shall provide offline mode for saved resource data, allowing users to access shelter and food bank information without an internet connection.

Priority 3 – Nice-to-Have (Future Enhancements)

7. Advanced Search & Personalization

- 7.1 The system shall provide a search history feature for users.
- 7.2 The system shall allow users to filter search results by resource type, availability, and distance.
- 7.3 The system shall allow users to share resource information via social media or email.

8. Data Tracking & Analytics

- 8.1 The system shall allow users to view historical data on resource availability.
- 8.2 The system shall allow relief workers to generate reports on resource shortages and availability.
- 8.3 The system shall allow system administrators to generate reports on system usage.
- 8.4 The system shall enable system administrators to manage database backups and monitor performance.

9. Team Collaboration & Accessibility

- 9.1 The system shall enable relief workers to communicate with each other via an internal chat system.
- 9.2 The system shall provide multilingual support for users in disaster-prone regions.
- 9.3 The system shall provide accessibility features, including screen readers, high-contrast modes, and text resizing options.

10. Security & Performance Enhancements

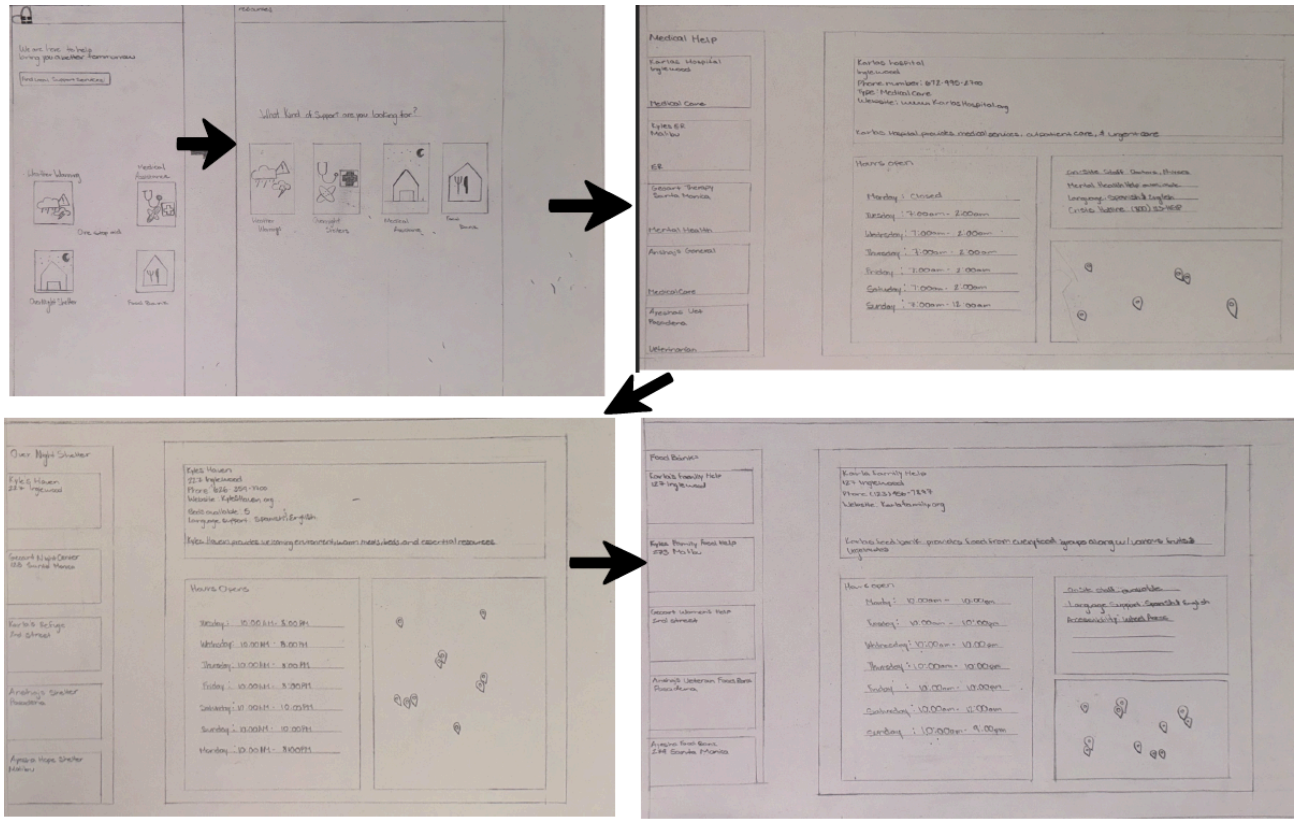
- 10.1 The system shall allow system administrators to update the system with new features.
- 10.2 The system shall enable system administrators to manage system maintenance schedules.
- 10.3 The system shall allow system administrators to monitor and manage API usage.
- 10.4 The system shall allow system administrators to manage third-party integrations.

Summary of Changes from Milestone 1

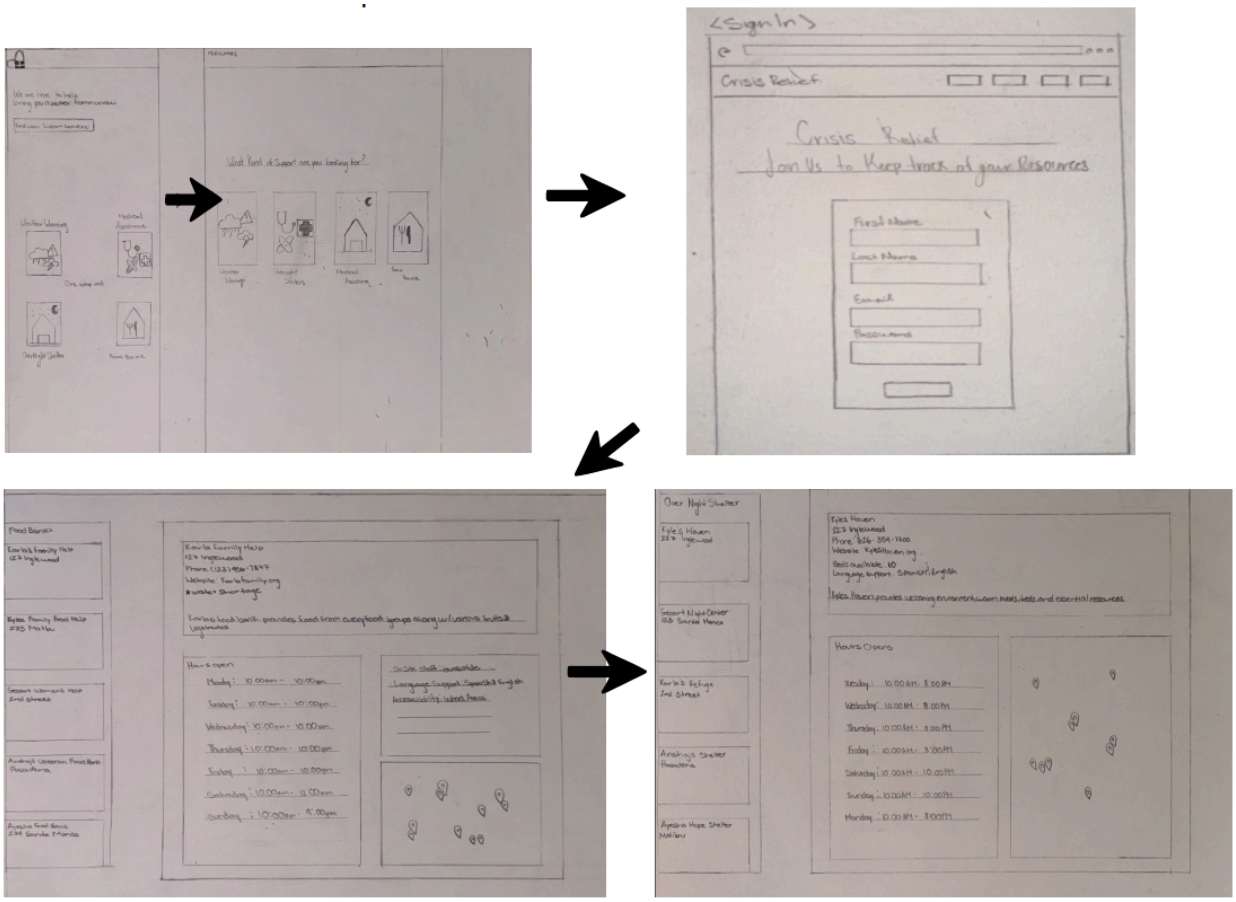
1. Reprioritized Features
    - Priority 1 now focuses on API integrations and real-time data (shelter availability, food bank stock, weather alerts).
    - Priority 2 focuses on user-generated data and mobile optimization.
    - Priority 3 includes advanced analytics, communication, and extra security features.
    - Shortened and combined so we don’t have as many as before

Mockups/Storyboards

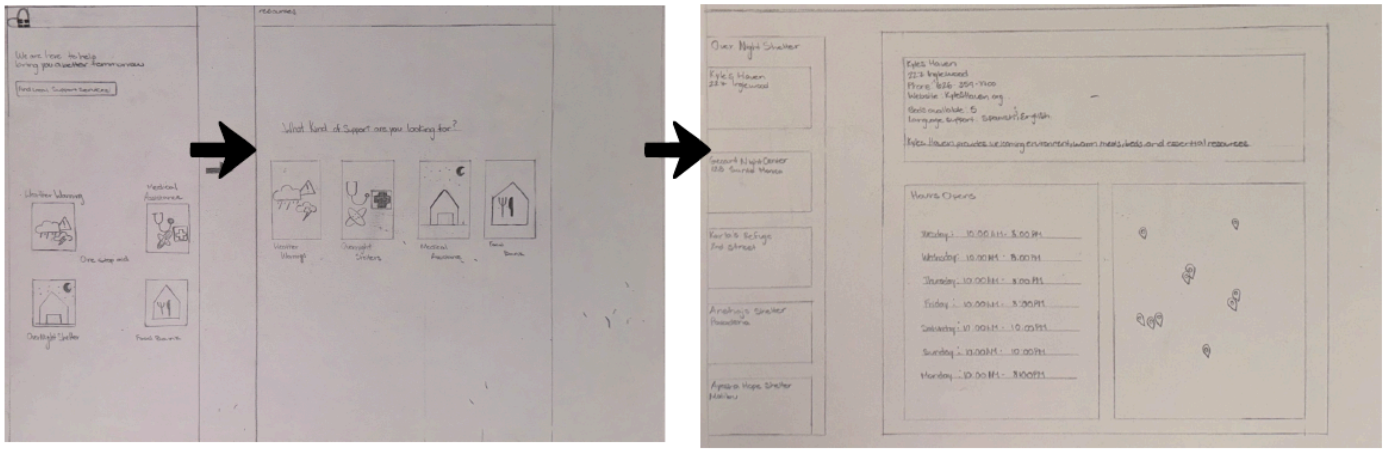
Use Case 1: Finding Emergency Resources Mockup



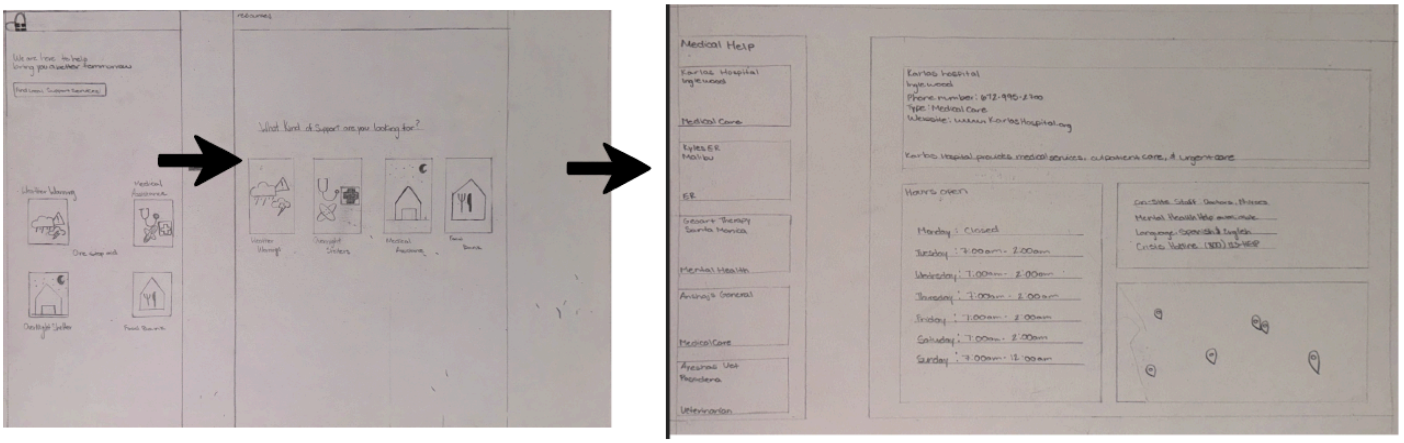
Use Case 2: Real Time Updates Mockup



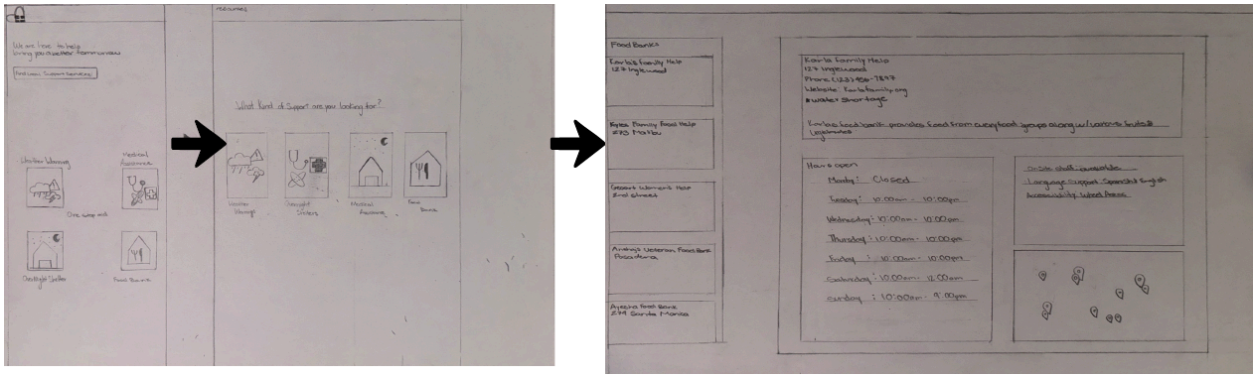
Use Case 3: Shelter Locator Mockup



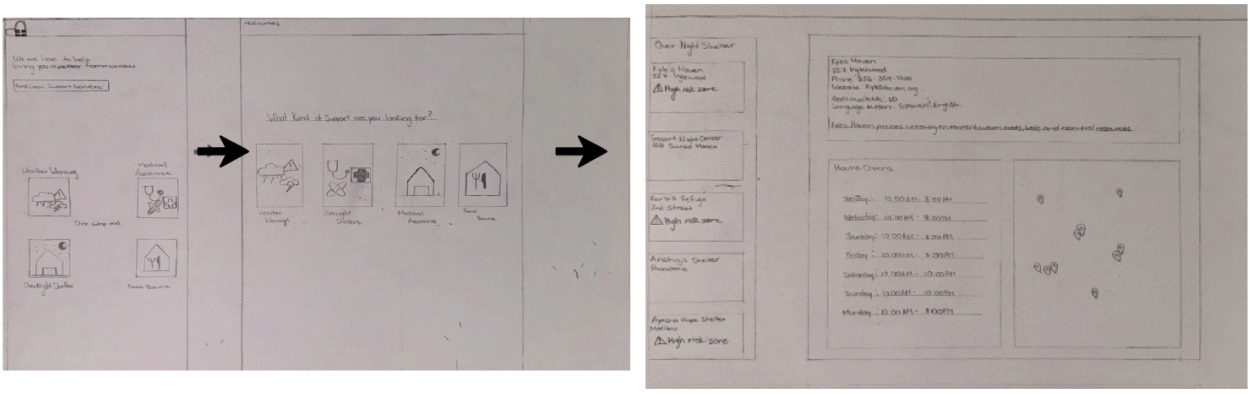
Use Case 4: Medical Assistance Mockup



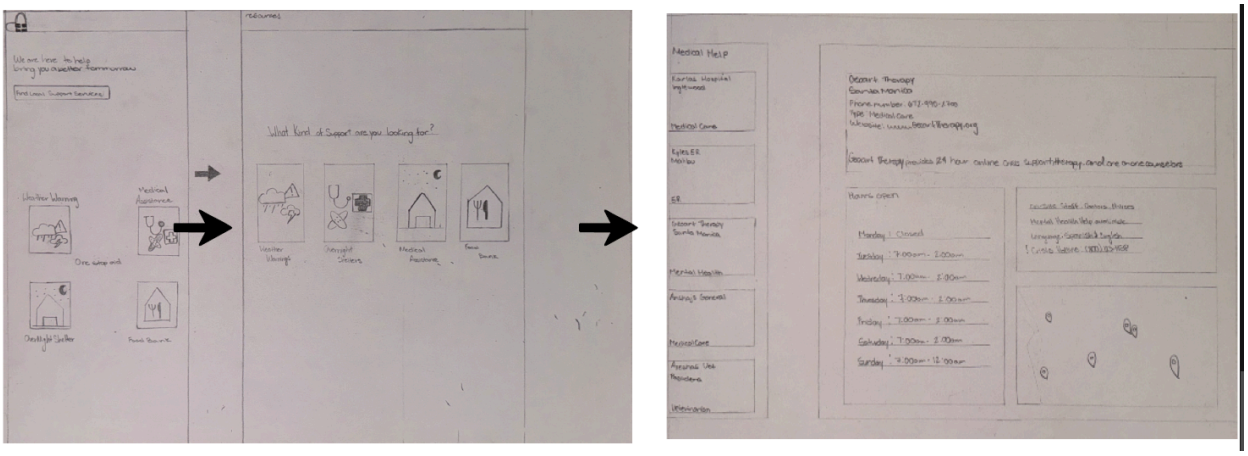
Use Case 5: Real-Time Weather Warnings on Resource Pages Mockup



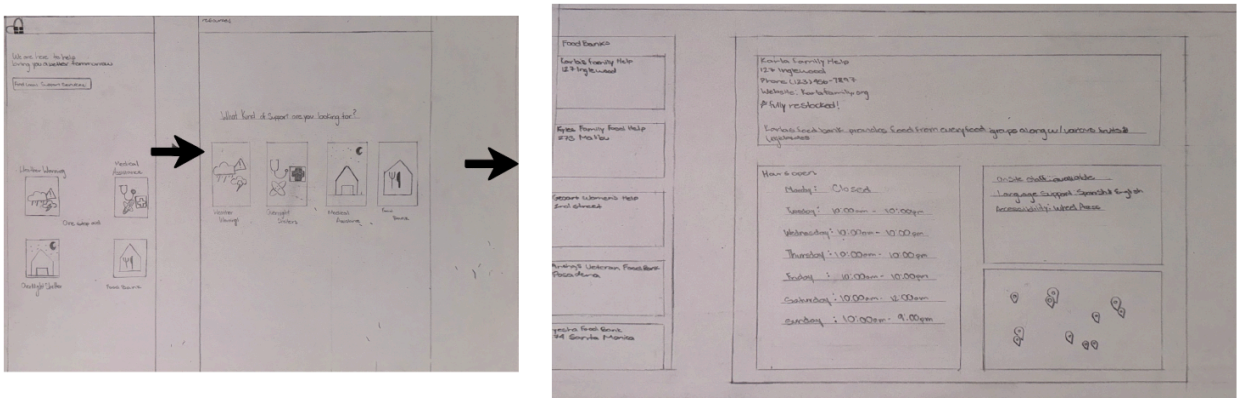
Use Case 6: Identifying High Risk Areas and Safe Zones Mockup



Use Case 7: Mental Health Support Review Mockup



Use Case 8: Food Bank Real-Time Update Mockup



High-Level Database Architecture

- Workers
    - The database shall allow workers to update the stock of locations.
    - The database shall allow workers to update only their places of work.
    - The database shall allow workers to create high risk areas and safe zones.
- Locations
    - The database shall return details of locations such as address, contact info, and operating hours to anyone who requests.
    - The database shall store the history of the location's resources.
    - The database shall store reviews of locations.
    - The database shall allow anyone to view reviews.
    - The database shall allow users to submit reviews.
- Users
    - The database shall store users' saved locations.
    - The user shall only have access to their own saved locations.
    - The database shall store users' search history.
    - The user shall only have access to their own history.
    - Database shall store sessions tokens for users that will only last 5 hours.
- Administration
    - The database shall store user roles and permissions.
    - The database shall allow system administrators to configure only their user roles and permissions.

DBMS Selection

For our database management system we will be using PostgreSQL, as it has more features and is able to do more complex tasks.

Database Organization

- Users
    - Attributes
        - Password, User ID (Primary key), Address, Search History, Name
    - Relationships
        - Writes Reviews, in a one to many relationship
        - Has sessions, in a one to one relationship
        - Saves locations in a many to many relationship
- Workers
    - Is a user
    - Relationships
        - Updates locations, in a many to many relationship
        - Updates High Risk Areas in a many to many relationship
- Session
    - Attributes
        - Session ID (Primary key), User (Foreign key)
    - Relationships
        - Belong to a user in a one to one relationship

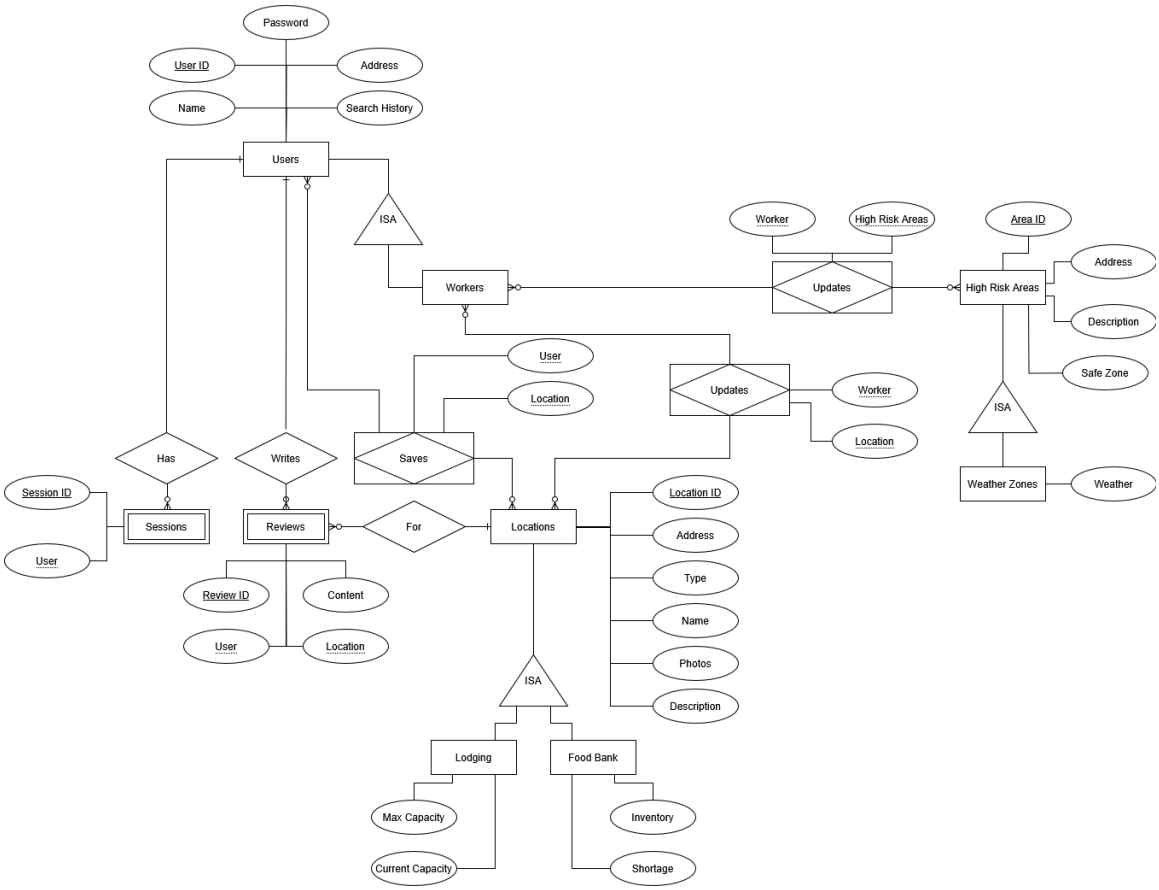
- Reviews
    - Attributes
        - Review ID (Primary key), User (Foreign key), Location (Foreign Key), content
    - Relationship
        - Belongs to the user in a one to many relationship
        - About locations in a many to one relationship
- Locations
    - Attributes
        - Location ID (Primary Key), Address, Types, Name, Photos, Description
    - Relationship
        - Have reviews in a one to many relationship
        - Updated by workers in a many to many relationship
        - Saved by users in a many to many relationship
- Lodging
    - Is a locations
    - Attributes
        - Max capacity, Current Capacity
- Food bank
    - Is a location
    - Attributes
        - Inventory, Shortage

- High Risk Areas
    - Attributes
        - Area ID (Primary Key), Address, Description, safe zone
    - Relationships
        - Updated by workers in a many to many relationship
- Weather zones
    - Is a High Risk area
    - Attributes
        - Weather

Media Storage

We shall store the images and review contents inside the database as BLOBs.

Entity Diagram



Backend Architecture

We have a backend that is based on microservices, i.e., different parts of the system are independent of each other but work well together. This is more scalable, fault-tolerant, and maintainable. Instead of a single large backend, we have two primary backend servers with separate tasks.

Backend Server 1 performs Authentication & Resource Management, granting secure access to users and processing all the data related to shelters, food banks, and hospitals. Backend Server 2 performs Real-time Alerts & WebSockets, offering real-time notifications to users about emergency situations. Both servers communicate with the frontend and database via APIs and WebSockets to enable a smooth transfer of data across the system.

Backend Services and Responsibilities

Each backend service performs a specific function in processing multiple parts of the system. Backend Server 1 handles user authentication and control of required resources. Auth Service ensures users are securely authenticated by JWT (JSON Web Token) authentication and passwords are securely encrypted by bcrypt hashing. It further controls access to different features based on roles. Resource Management Service manages hospital data, food banks, and shelters. It facilitates Create, Read, Update, and Delete operations, persists data in a sharded MySQL database, and uses Redis to cache frequently read data for improved performance.

Backend Server 2 is tasked with real-time communication and notifications. The Alert Service notifies users in real-time of emergency updates, such as a shelter reaching capacity or an area

becoming dangerous. Redis is used to buffer alert messages temporarily before forwarding them to users to improve efficiency. The WebSocket Service enables live interaction, delivering real-time updates to users without requiring them to refresh their application. This service constantly awaits changes to the database and broadcasts updates through WebSockets (ws://). Organizing the backend in individual services ensures each component functions well while allowing for future expansion without major disruptions.

Scalability & Performance Considerations

To address increasing user load and ensure system stability, we have employed some mechanisms for scalability. MySQL database is sharded in a way that we do not have one large database but split data into smaller, easier-to-handle pieces. Shard 1 holds user data, while Shard 2 holds resource-related data. This reduces database overload and allows faster execution of queries.

Aside from that, we leverage read replicas for database load balancing. Instead of directing all queries to one primary database, we send read-heavy requests to read replicas to improve system responsiveness. Redis caching also improves performance by storing frequently requested data, such as shelter locations and resource status, to reduce unnecessary database queries.

To ensure easy scalability, horizontal scaling is supported in our backend, i.e., if traffic increases, we can deploy additional instances of Backend Server 1 and Backend Server 2 to handle more requests. This distributed system makes our system responsive, quick, and accessible even under high traffic.

Security Considerations

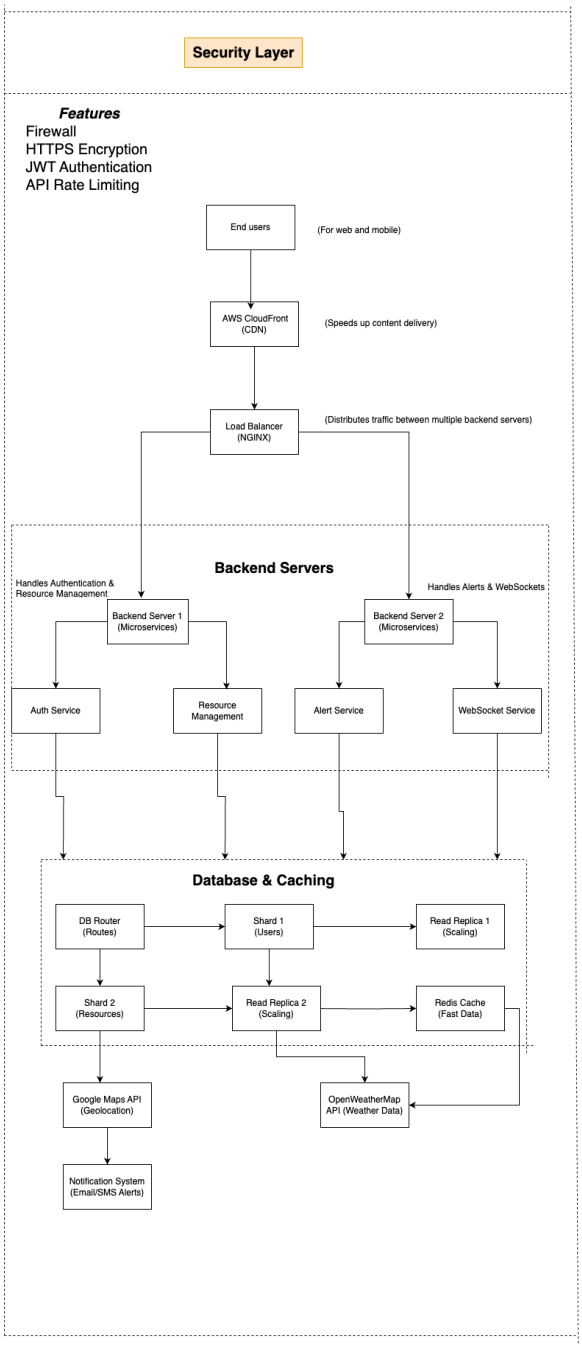
Security is given utmost priority in our backend design for the protection of the user data as well as from unauthorized access. The system follows JWT authentication for ensuring a safe session of users while accessing secured resources. API requests are protected by API Rate Limiting to avoid sending a large number of requests from a single user to avoid potential abuse as well as DDoS attacks.

All backend communications are HTTPS-encrypted, safeguarding data transfer between servers and clients. Firewalls and network limitations also ensure that only valid requests are permitted to access our backend services. These measures work together to safeguard sensitive user information and maintain system integrity.

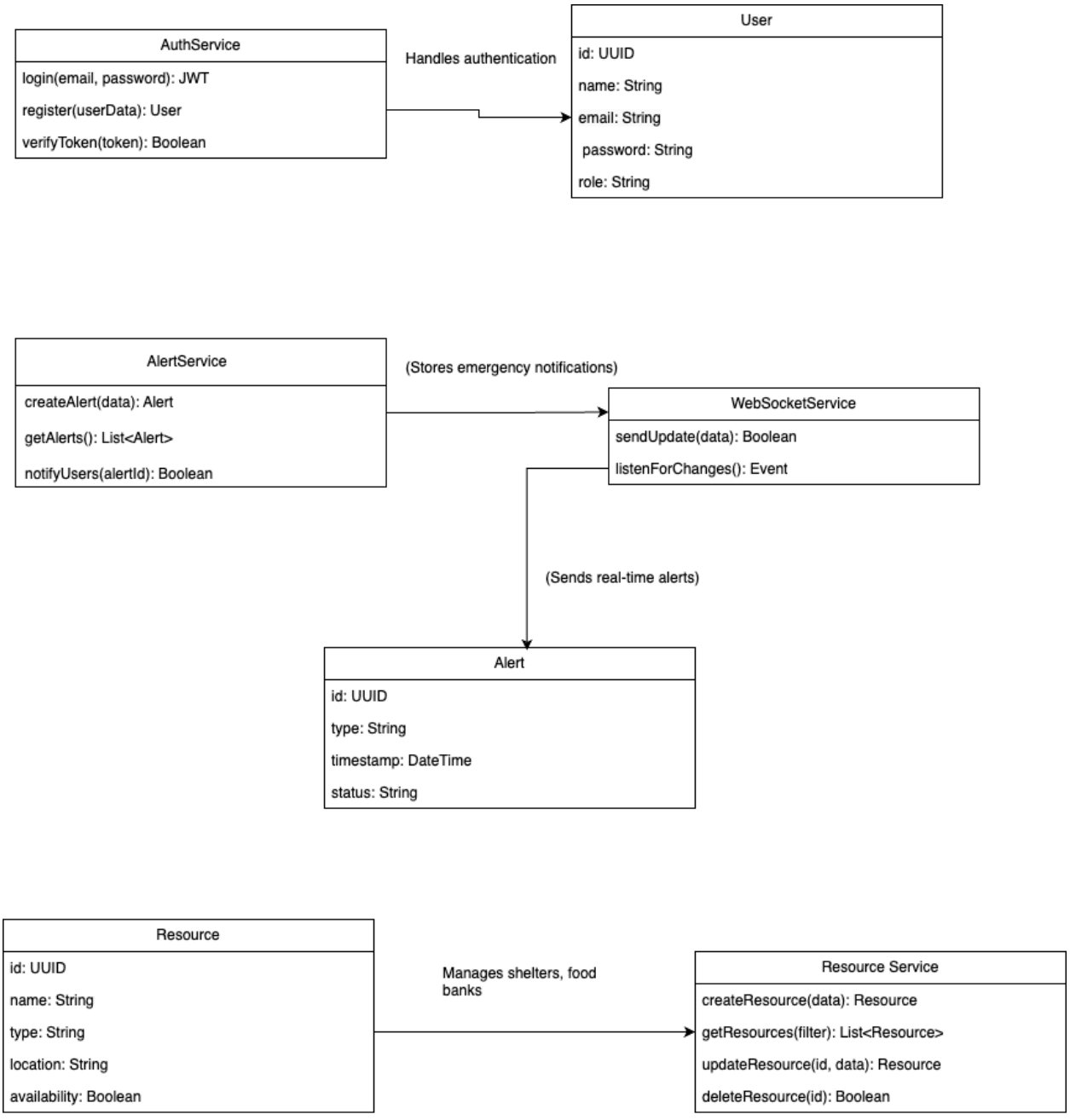
UML Class Diagram

The UML Class Diagram represents the backend structure on the code level, breaking down how different elements communicate with each other internally. Every service, i.e., Auth Service, Resource Service, Alert Service, and WebSocket Service, is represented as a unique class. All these services talk to database models like User, Resource, and Alert, write and read accordingly. Additionally, event-driven logic and WebSockets are executed through a custom WebSocketHandler class to ensure efficient handling of real-time updates. Through the designing of the system along different object-oriented principles, we ensure the code is maintainable, reusable, and modular.

Scalability Diagrams:

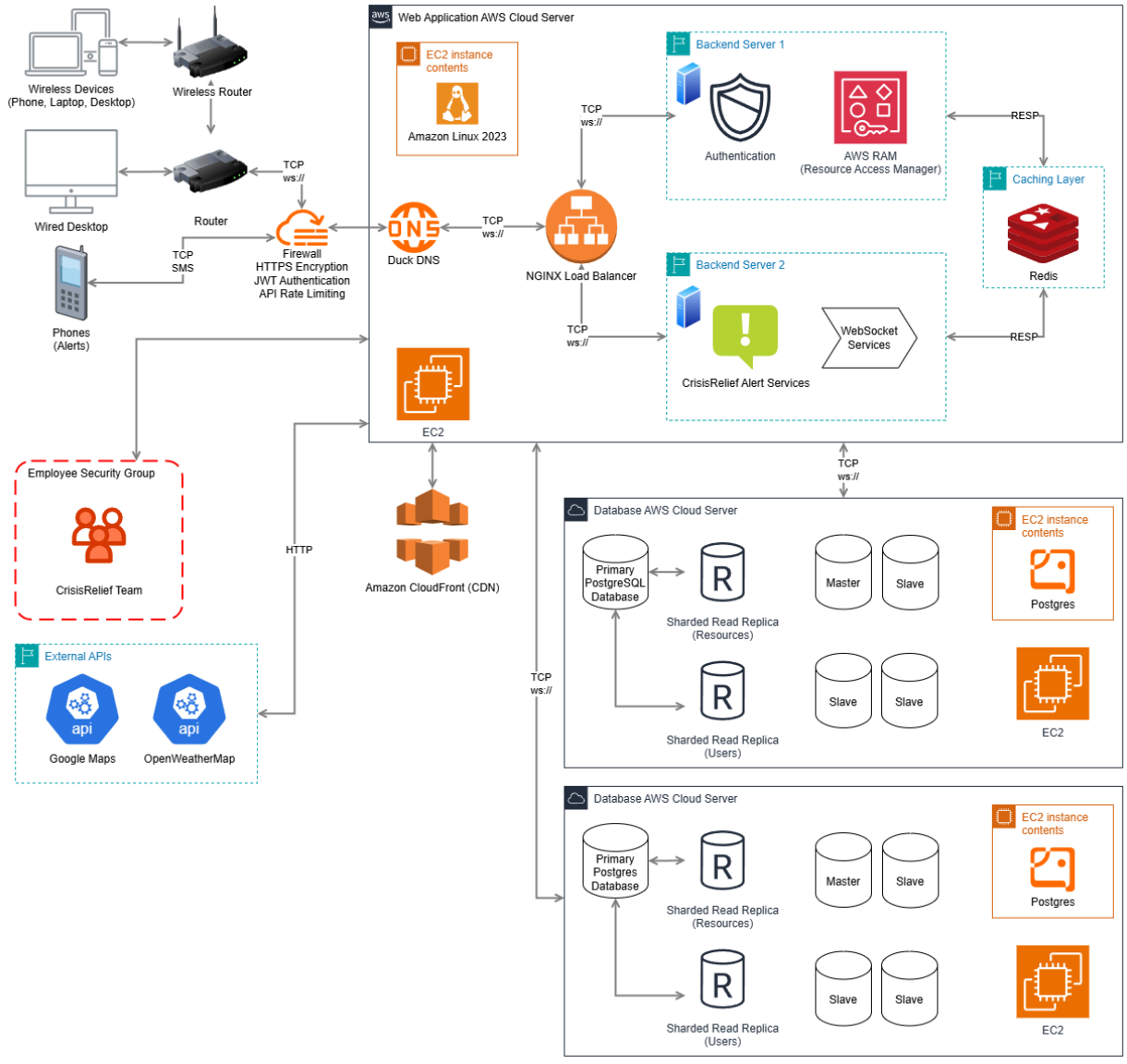


UML Class Diagram

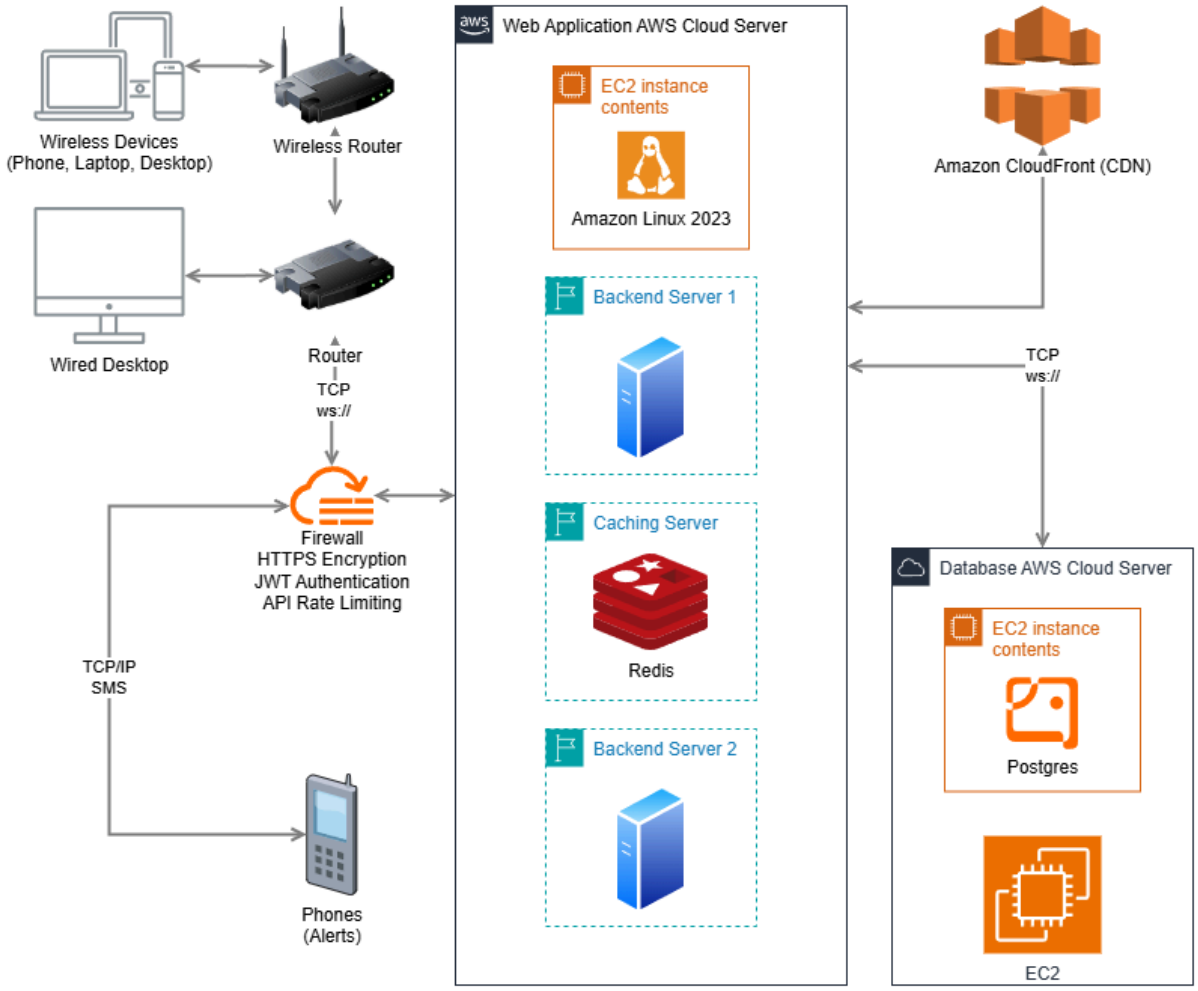


High-Level Application Network Protocols and Deployment Design

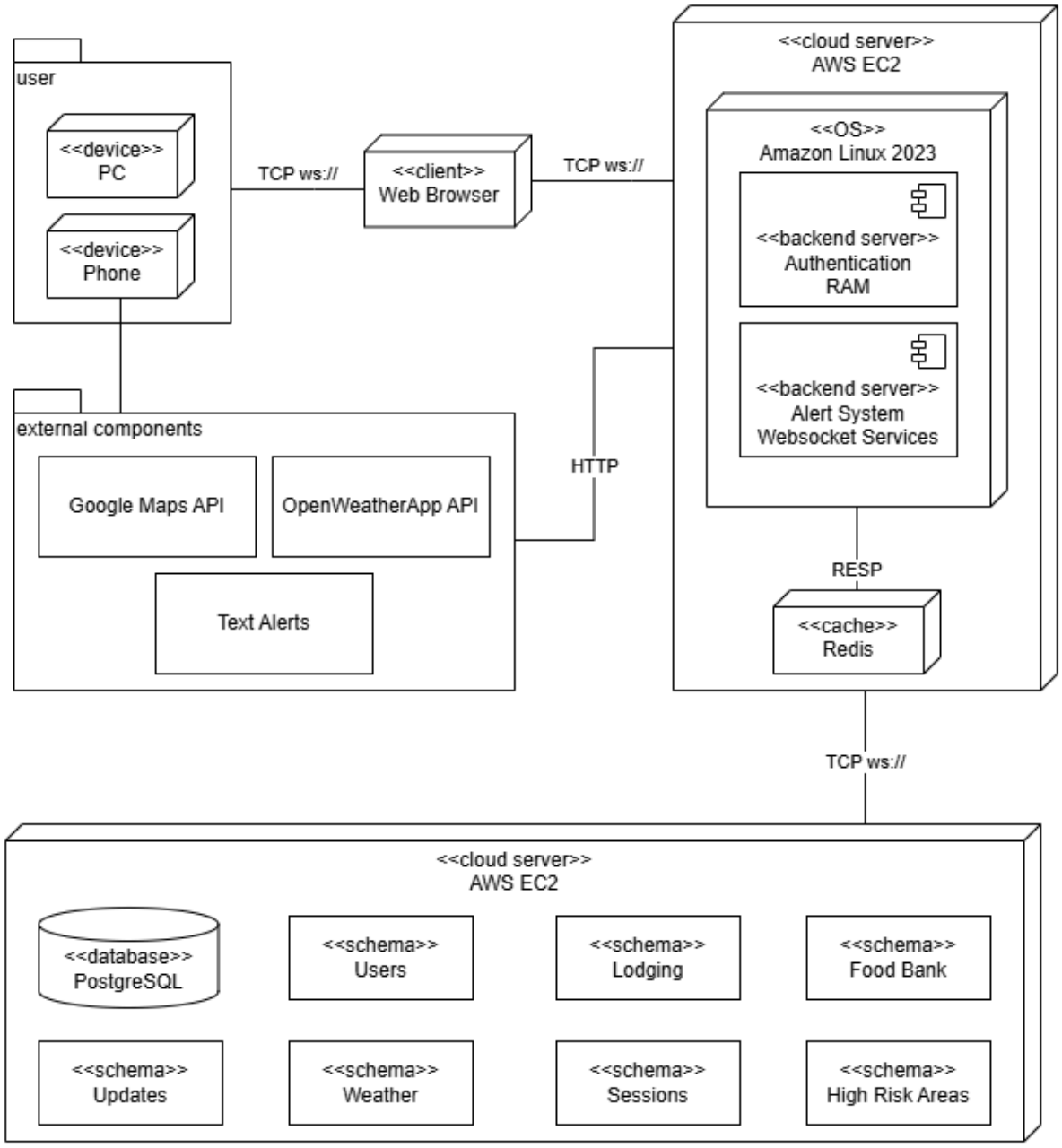
Network and Deployment Diagrams



Application Networks Diagram



Deployment Diagram



High-Level APIs and Main Algorithms

5.4 High-Level APIs

The Crisis Relief System exposes a set of RESTful APIs and WebSocket endpoints that support user authentication, resource management, emergency alerts, real-time notifications, and external API integrations. These APIs are designed to ensure scalability, security, and efficiency, following best practices such as rate limiting, caching, pagination, and event-driven messaging (Kafka Queue).

1. Authentication APIs (AuthService)

These APIs handle user login, registration, and session management with JWT authentication for secure access.

- POST /auth/login – Authenticates users and returns a JWT token.
- POST /auth/register – Registers a new user in the system.
- GET /auth/verifyToken – Validates the user's JWT token for authentication.

2. Resource Management APIs (ResourceService)

These APIs allow users to search, add, update, and delete crisis relief resources like shelters, food banks, and hospitals.

- GET /resources – Retrieves available shelters, food banks, and hospitals.
- GET /resources/{id} – Fetches details of a specific resource by ID.
- POST /resources – Creates a new shelter or resource entry.
- PUT /resources/{id} – Updates details of an existing resource.
- DELETE /resources/{id} – Removes a resource from the system.

Additional Features for Optimization:

- GET /resources?type=shelter&location=SanJose – Filters shelters in a specific city.
- GET /resources?availability=true – Returns only available shelters.
- GET /resources?page=1&limit=10 – Implements pagination for better performance.

3. Emergency Alert APIs (AlertService)

These APIs manage emergency alerts, ensuring users receive real-time notifications about crises.

- POST /alerts – Creates a new emergency alert in the system.
- GET /alerts – Retrieves all active emergency alerts.
- GET /alerts/{id} – Fetches a specific alert by ID.
- POST /alerts/{id}/notify – Sends an emergency notification to users via SMS and email.

4. Real-Time Communication APIs (WebSocketService)

WebSockets enable instant, real-time updates on shelter availability and emergency alerts.

- WebSocket Endpoint: ws://alerts – Establishes a WebSocket connection for real-time updates.
- sendUpdate(data) – Pushes live alerts to all connected users.
- listenForChanges() – Listens for updates from the backend and delivers instant notifications to users.

5. Event-Driven Messaging APIs (Kafka Queue)

The system uses Kafka Queue to process real-time events and ensure scalability in delivering notifications.

- publishAlert(alertData) – Sends a new emergency event to Kafka for processing.
- consumeAlerts() – Listens for new alert events and triggers notification broadcasts.

6. External API Integrations

The Crisis Relief System integrates third-party APIs to enhance functionality and provide users with accurate location and weather data.

- Google Maps API (GET /maps/geolocation) – Fetches geolocation data for nearby shelters.
- OpenWeatherMap API (GET /weather/alerts) – Retrieves weather-related crisis alerts.
- AWS S3 API (POST /s3/upload) – Uploads and stores shelter images and multimedia files.

7. Notification APIs (NotificationSystem)

These APIs send email and SMS notifications to users when an emergency occurs.

- POST /notifications/email – Sends email alerts to users.
- POST /notifications/sms – Sends SMS notifications for emergencies.
- GET /notifications/history – Retrieves past notification logs for audit purposes.

8. Admin & Monitoring APIs (Admin Dashboard)

For system administrators, these APIs allow monitoring, approving, and managing platform data.

- POST /admin/resources/approve/{id} – Approves newly added shelters before listing.
- DELETE /admin/alerts/{id} – Removes an outdated or false emergency alert.
- GET /admin/logs – Retrieves system logs for security and performance monitoring.

9. System Logging & Error Handling APIs

These APIs track API performance, errors, and security events.

- GET /logs/errors – Retrieves a list of API errors and failures.
- GET /logs/usage – Fetches API usage metrics for monitoring and debugging.

10. Performance Optimization Features

To enhance system efficiency, the following optimizations are included:

- Rate Limiting – Prevents excessive API requests per user.
- Caching (Redis Cache) – Stores frequently accessed data for faster retrieval.
- Pagination – Limits large query responses to improve API speed.

Advanced Search

Depending on a user's interest or needs, CrisisRelief will be able to provide catered results that will help a specific user. A search will filter through key terms such as the main groupings of aid while progressively getting more specific with more and more input. For instance, a user can include "Food Bank" and "San Francisco" in their query. As a result, CrisisRelief will read from the Food Bank database for any entries that belong in San Francisco. In addition to user-provided input, CrisisRelief will take advantage of user location and user history when permitted as to avoid irrelevant responses.

Strict filtering may pose a greater obstacle in situations where aid is of need. It will be important for our search to have leniency in the input provided. Spelling errors or generalities should be treated the same as an identical match.

Ranking

A key factor in an aid's convenience is accessibility. If it's inaccessible, it should not be treated with any importance. In terms of ranking, distance will typically serve as a good default for what should be displayed to a user first. Using the Google Maps API alongside stored information on user location will provide such metrics. However, there are other things that are important to also keep note of. High-risk zones or emergencies due to circumstance will be considered thanks to the up-to-date nature.

For instance:

- Food Bank A is a 45-minute walk away but is dealing with flooding
- Food Bank B is an hour's walk away but has no obstruction

Food Bank B will be the one prioritized.

Community input is valued for CrisisRelief, so relevance will alter results based on reviews and reputation. Knowing where to avoid is especially important for things that may be costly or time-consuming.

Key Project Risks

- Skills risks
    - Risk: Some team members are inexperienced in TypeScript, React, or Node.js.
    - How to fix: Everyone is working to make up for what they are inexperienced in, either through watching YouTube videos to help with specific tasks, or asking others that are more well-versed in the subject.
    - Risk: Backend team is newer to MySQL database management.
    - How to fix: The entire backend team is working together to make sure everything is well and helping in spots which the others may not understand.
- Schedule Risks
    - Risk: Database was not set up towards the end of Milestone 1
        - Fix: Francis and Karla were able to work together to set up the database with the slightly extended deadline. Amazing work.
    - Risk: Potential delays in integrating Google Maps API and OpenWeatherMap API.
        - Fix: Assign a dedicated team member to API research and testing early to prevent bottlenecks.
- Technical Risks
    - Risk: Potential server downtime or scalability issues due to high traffic.
        - Deploy on AWS EC2 with auto-scaling, use Docker containers for easy deployment, and implement load balancing

- Teamwork Risks
    - Risks: Poor communication may slow progress, especially across different roles (frontend, backend, database).
        - Fix: We are meeting three times a week (T,Th,Sat) to see how everyone is doing. We also have a notion set up to see how tasks are going along, as well as a team leader that is great at checking in with us.
    - Risk: Differences in coding styles leading to merge conflicts.
        - Fix: Possibly establish a code formatting guideline, such as Prettier for TypeScript, and check in with everyone before merging pull request
- Legal/Contents risks
    - Risk: Licensing issues with third-party APIs (Google Maps, OpenWeatherMap).
        - Fix: Verify API terms of use and use only free-tier options unless necessary to upgrade. Monitor API rate limits to prevent unexpected costs.

Project Management

How we are going to be managing our project is through the use of Notion. Our Notion setup is pretty simple. We have two separate dashboards, one for checkpoint one and one for checkpoint two. Both dashboards include basically the same things. They include the tasks, the description of the task that was given to us on GitHub, who is assigned that task, and when the task should be due. Anyone can move the task to currently in progress or completed. Sadly, our task management wasn't up to par with how we expected this milestone. We shall be using Notion still, but will need to do better to help everyone meet their deadlines, by either checking up on one another more and/or keeping ourselves accountable.

Invitation Link:

https://www.notion.so/Crisis-Relief-Milestone-3-1a8617e5d24380f09984eb7399cd92ab?pvs=4

Current Searchable Terms

- **Locations**:
  - “shelter”
  - “food bank”
  - “Main City Shelter”
  - “Downtown Emergency Shelter”
  - “Suburban Family Shelter”
  - “Community Food Bank”
  - “Downtown Food Hub”
- **Reviews**:
  - “helpful”
  - “excellent”
  - “clean and well-organized”
  - “nutritious options”
  - “safe haven”
- **Search History:**
  - “food bank services”
  - “emergency shelter”
  - “downtown shelter”

List of Team Contributions (Team Lead Only):

Team Evaluation Feedback

I am pleased to share that we were able to achieve the milestone.

Geoart Corral (10/10): Geoart was an excellent team member. His prompt responses and commitment to the project were greatly appreciated. He put a lot of effort into the search algorithm and also improved checkpoint 1 documentation.

Ayesha Irum (10/10): Ayesha put in a lot of effort to complete the diagrams for checkpoint 1 and remained in contact with the team. Her contribution in team discussions and on the rating and ranking sections was really helpful.

Kyle Nguyen (10/10): Kyle coordinated our documents and collaborated well with Geoart with respect to the Search Bar UI. His documentation was detailed and focused, and this ensured our project documents were organized.

Karla Cardenas Andrade (8/10): Karla was excellent in the storyboard. At times, however, her slow responses on Discord created additional work for other team members. Addressing this communication issue would allow everyone to work together more effectively in the future.

Francis Aviles (9/10): Francis assisted greatly with the database and ensured everything was properly configured. Although he performed well with the technical aspect, he did not meet the team's timelines, which put the entire project behind schedule. It would greatly benefit him if he could enhance his time management and discuss the schedule expectations.

Background Reading

Class Notes - Geoart Corral

Continued on next page…

CS

Week 4

Human Interaction, Network, Databases, CG?
Network Security
Cyber Security → Specialize in 1 thing
Blend areas

MVP = Minimum Viable Product

Look Mobile for V2
Copy V2, not V1

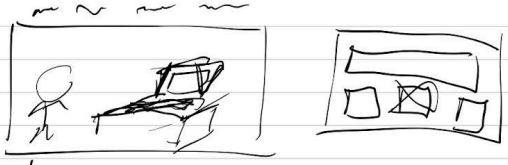
M2

3) Improve the data definitions
4) Prioritize High-Level Functional Requirements
Priority 3 ex = Payment System

Priority One: → Done as a whole team
1. User
1.1 ...
1.2 ...
2. Reg User
2.1 ...
2.2 ...
Priority Two

5) Mockups / Storyboards
· Mockups for all use cases
★ Needs to be done by hand
· Doesn't have to be crazy either

Mockup: → Description



wireframe (Point to class mockup)

6) High-Level System Design
· TCP = Protocol
· Guarantees receiving a packet from a transmission
· Ensuring security (Netflix, Youtube)
UDP = Losing a packet = no transmission; Faster
Packet loss, Lost

6.2) Backend Architecture
Caching, Containers, etc.
|
UML Class Diagrams
(OOP or TCP Protocol stands Programming)

6.1) Databases
...

6.3) High Level Approaches Network Protocols and Deployment Design
· Not required to implement new protocols but recommended
· APIs typically taken care of 9[illegible]

6.4) High Level APIs and Math Algorithms
· When APIs to use
· Deal with about Post, Get, etc.; use plain english

7) Checklist

!

8) Project Management

Checkpoint #2

- Vertical Prototype; built upon for rest of project

!

VP: Connectivity

- Must validate encryption

VP: Algorithms

- Secure Functionality, not geared for UX
- Pairing and Pairing
- Optimization

Front End?

- ~~X~~ Proof of inserting into database! Create an account
- ~~X~~ Can pull data from database! Search Bar
- ~~X~~ Test Algorithms
  - If not found, suggest a recalibration

Week 5 Software Processes and Introduction to Extreme Programming 2/27/25

★ No class Mar 13

★ Next week: CI/CD = Continuous Integration
- Workflow Actions

- Only fix M1, don't add
- No carrying variables into other milestones

Business → Requirements ✗ → Design Architecture → Implementation → Testing Integration → Maintenance

Marketing →

Software Processes

Waterfall

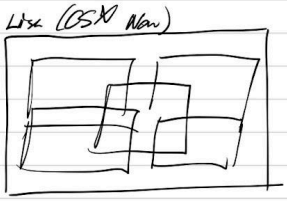
- One class

- Cannot move backwards
- Requirements => Design Arch. Not for this class
- Everything flows in updates
- Why can't you go back?
  - You're working with other teams
  - They all agree first
  - Marketing makes social media post, can't drop everything
- ★ Discipline

Brief History

- Lisa OS → Macintosh
  - Used this model
- User give feedback now
- Windows 93-95, they didn't use user feedback; took 3 YEARS
  - Needed faster turnarounds
- Startups use "waterfall" to ensure they can please investors
- Poly corporations use a mix

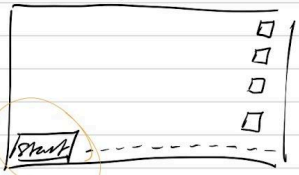
Lisa (OSX Now)



- Windows everywhere
- Hard to manage

~Why no Usability?
- Didn't think about Usability until 1997

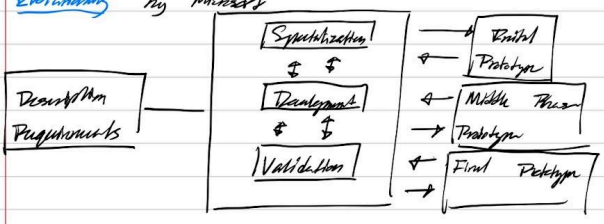
Windows 95



Final Exam Question ★ ★
- Maintenance Lasts the longest (forever)
- Most controversial and gives most trouble? Implementation

- Start of microsoft monopoly
- ★ Usability is important

Evolutionary by Microsoft

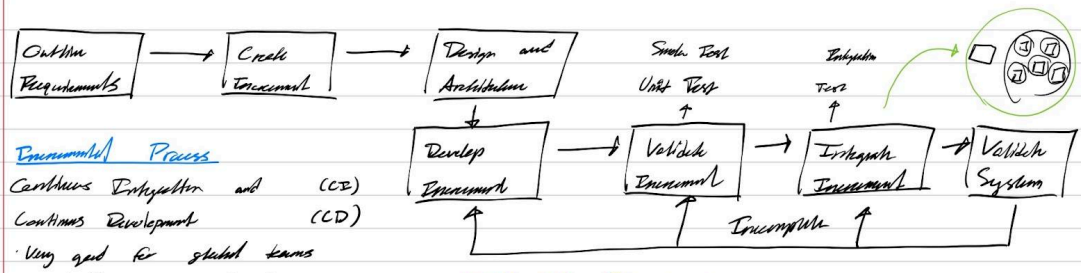


- Synchronous; Not linear

Disadvantages
- Cost increase exponentially
  - Prototypes take a lot of time
- Created a lot of spaghetti code

~What would a middle sized company use?
- Both!

- CI/CD = Continuous Integration / Continuous Development



Incremental Process

Combines Interpolation and (CI)
Continuous Development (CD)

- Very good for global teams
  - Always team to test
- All components are decoupled
- Must validate everything at the end
- Used by today's industry

Important for career

Disadvantage

- Doesn't merge the team
- No prototypes

Advantage

- Process is good for product

Best?

Incremental with CI/CD
Extreme Programming (XP)
User Center Design (UCD)

- Code review
- Pair Programming
  - Water while code; suggest changes when observing
- CD/CI
- Agile
  - Sprints = Checkpoints
  - Scrums : Team lead brings knowledge and asks questions about everything
  - Dependencies : Tasks/Deadlines from team lead

Group Work vs Team Work

Group Work = Ideas → Individual
Team Work = Constant communication and synchronization

CONGRATS TEAM! We won boat MI!

Lecture Again

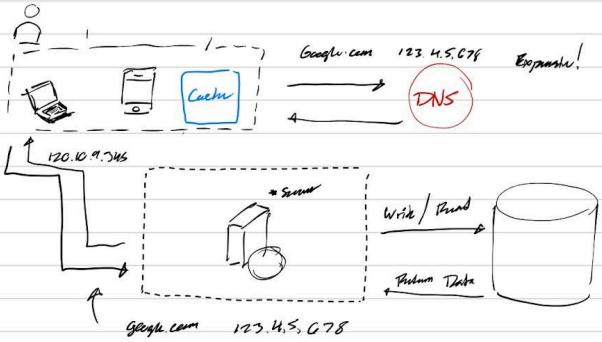
- Specify older versions for non-functional requirements
- Competitor analysis = Invents the unique features based on what was approved
- "Take the class seriously
- No AI...
  - ✗ Copy and pasting
  - ✓ Use for productivity
- Team leads Guide and check for high quality

Week 6 System Design Part 3/16/25

Diagrams

* How many devices are we supporting?
  - Specify specific devices
  - 'Ex' * Can do it however we like

- Point to point? What protocol?



DNS

- Interacts with the client server
- For ex, Imagine user can
  - Bo) jortiz@stan.edu
    - Domain
- Must turn an IP Address
  - Usually changing
- DNS = Hash table of IP Address (key) and Domain (value)
  - Can turn many domains
- Need cache, otherwise expensive

3 - Databases

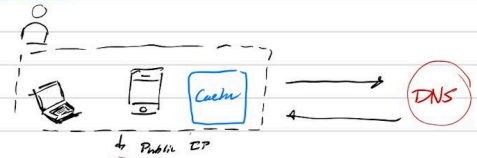
Relational

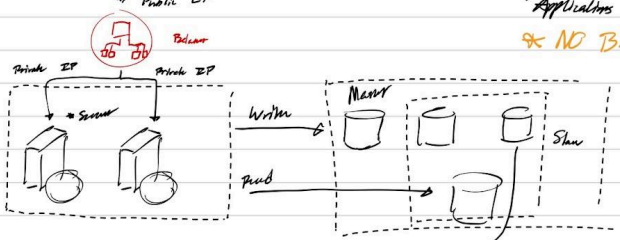
- Store and persist but slower
- Good for truth
- All or nothing → Decreases performance
- Need to know anything on entities
- Datatypes decided upon creation

Non-relational (NoSQL)

- Low latency, very fast!
- Unstructured data
- Must sustain (Lifespan not stored)
- Memory limits

4 Balancers





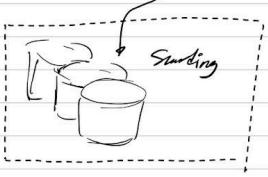
- Works like a VPN
- Very important
- What if a server was almost overloaded
- Automatically create virtual instances
- Applications like "Bezpit" use auto feature on balancers

* NO BALANCERS CAUSE IT'S EXPENSIVE

Master/Slave Protocol

Master = Write

Slave = Many read



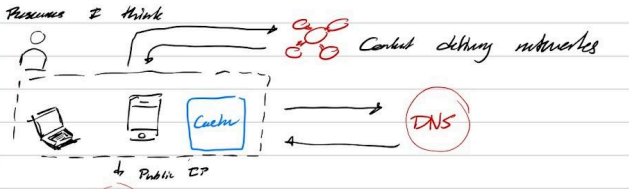
(?)

Standing = technique to reduce delay

Does it make sense to put a cache at server?

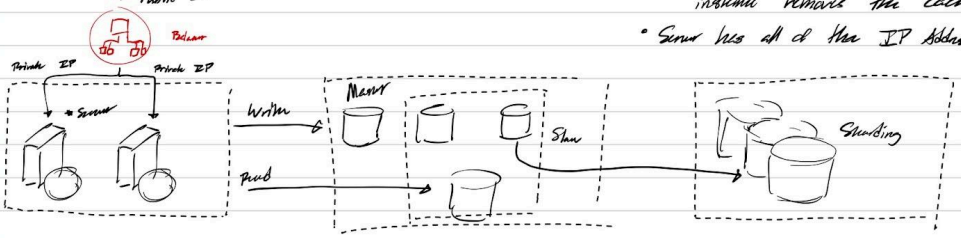
Yes! Redis is good for data; acts too

Works wrong with lots of caches

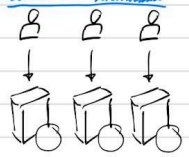


Content delivery networks

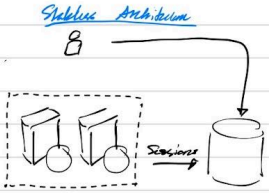
- Geographically located
- Per shall not being modified
- Don't need to cache stuff dealt with the CDN
- Caching a server means closely an instance removes the cache
- Server has all of their IP Addresses



Stateful Architecture



Stateless Architecture



- Saw the session in the database
- Requires Replication
- Stateless is good with backends

Data Centers

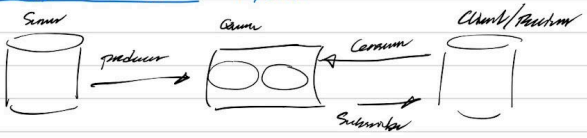
- Hold everything in multiple centers
- Very strong but very expensive

Message Queue

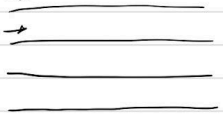
NoSQL = Not on SQL

- Many architectures together
- Less popularity because it can't solve complex problems

Message Queue Protocol Pub/Sub

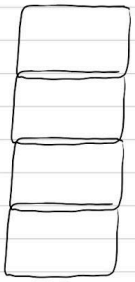


Channels Multiple Free

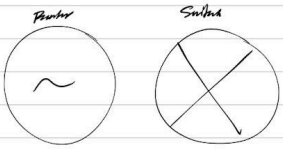


- Now can deal with HTTP protocol
- Visibles/Redis Feed API
  - Async w/ multiple channels

Monolithic Architecture



- Everything relies on the stuff above/below
- One failure is detrimental to the system
- Supports smaller user base
- Vertical vs Horizontal Scaling

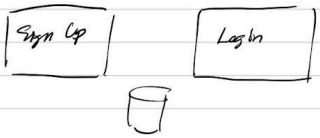


Not self Learning

Self learning

- Talks when the physical device can

Microservice Architecture



- Sharing databases requires keys
- Broker because one service can continue working
- Fault tolerance
  - Sequence Breaker
    - When component fails, the failure does not spread

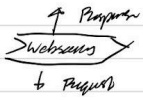
255.255.255.255 → Broadcasting

Week 7 - System Design Async Lecture Part 2 3/13/25

- Start with mapping out entities — Reference slides / PDF for diagrams
  - Encryption is important now

Monolithic Architecture - Small Scale 0-100 users ~19:40

- One thing wrong? Everything fails
- Database doesn't store media files
  - S3 Bucket Distribution
- Must encrypt client side if monolithic
- Websockets for messages
- DB has IP address to media files



Tech Stack Picks ~38:40

Frontend: React

Backend: FastAPI with python, Flask with python

- Async calls

DB ~39:50

- No SQL good for huge
- Can you define all entities? Yes — Need transactions? No — Store lots of data? Yes → No SQL — Scalable data? No
  - SQL database!
- Take note of scalability
- Monolithic favors SQL database
- Read queue can handle many requests

Conceptual Design of the database ~50:00

- Database entities (relations, strong / weak)
- Encryption and sessions = non-functional for security and auth

Entity Relationship Diagram (ERD) ~1:04:00

- Many to Many
- Associative Entity = relationship that becomes an entity
- Primary keys can't have dupes
- Attributes have no connectivity
- ISA = Can be one

Encryption + Networks ~1:22:20 — Be sure to reference the provided diagrams

- Application layer → Presentation → Sessions → Maintains sessions
  - Support Network Applications
  - HTTP, HTTPS, FTP, ...
- Client side: Encrypts when sending, decrypts when receiving
- TCP Protocol doesn't lose information
- UDP doesn't maintain a connection, it just sends it (real time)
- Modern apps use both
- Data Can in charge of router, protocol chosen by how big
  - Packets drop
  - Forwarding protocol = send packet then close connection

Link or Data Layer - Point-to-point protocol
  - Local networks and internet
  - Uses frames in the header
- Physical layer
  - Bits to the wire
- Router don't use TCP and UDP because of security breaches
- Conceptually, a monolithic layer

10-100 Users - Concurrent ~1:53:20

- Handled in parallel; synchronous
  - Bottlenecks will happen when waiting for things to finish
- Challenges: Message delays, latency, bottlenecking, DB gets overloaded, WebSockets consume memory, monolithic = no offline messages ... so relay load by letting do offline
- Improvements: 1) Auth service 2) Messaging Service (WebSockets and chat servers) 3) Assets from S3 instead Content delivery networks
  4) Firebase for notifications (move away from EC2 during Nodes

Optimize DB with Replication with Master-Slave technique

- Move request means your read a cache, like Redis
- Distributed websockets and can lead failures
  - Specific to this websocket example
- Message Queue stores messages offline as improve performance ~2:16:00 - Tech for this example

1K-10K Users ~2:22:00

Challenges: 1) Full Scale = Expensive! 2) Support millions of messages 3) Writes would be slow 4) Troubleshooting processes is hard
5) Encryption adds processing overhead 6) Media traffic increases bandwidth costs

Strategies: 1) Database Sharding 2) Event-Driven Architecture (Process message asynchronously, Kafka) 3) Improve handling (Redis)
4) Improve messages scalability via distribution 5) Improve latency with more caching

Week 8 Zoom Review, M3, and Database

Quiz Review

1) Key challenge of monolithic?
   - B) Inability to handle offline messages
2) Optimize database between 10-1000?
   - B) Master slave replication
3) Why Kafka and RabbitMQ during 1k-10k users
   - B) Offline messages => turns async to synchronous
4) CDN
   - B) Make traffic load and improve latency
5) Why monolithic to distributed
   - C) Load and scalability

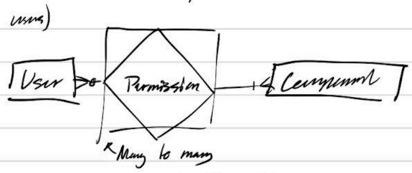
Milestone 3

- M2 Prototype doesn't need everything in the database
- Checkpoint 2 = Only priority 1! With **usability**
- Comment when not self explanatory

Permissions in DB (internal)

- Auth0 = authorizes with security stuff and stores a token in the machine (don't have to implement)
- Basic: User: one or more permissions

(0-10 users)



Users

uid	name
1	Jose
2	Cha
3	Bob

Permissions

token	pk pid	uid	time
1	1	1	—
2	1	2	—
3	2	1	—

pk component

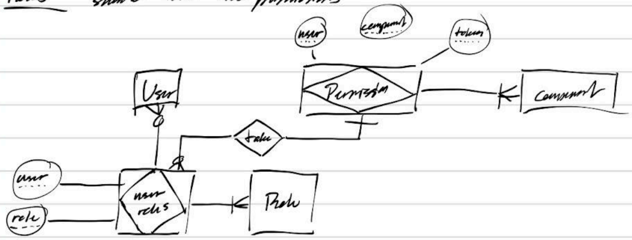
cid	desc
	images
	video
	calls

- Not good for big system

12 → Hashed → SHA256

- Becomes huge if more than 10
- **Roles** should have the permissions

We could do this if you want



Normalization
Required for this class

Normalization 7NF

- Decreases redundancy (related to good scalability)
- Increases integrity
- Sometimes decreases the performance of the database

Normalization + Optimization in order to succeed

|

1NF (First Normal Form)

- All the values in the same column are atomic
  - Only has 1 piece of data with a native datatype

|

- No repeated attributes in the same group
All tables must have a PK (primary key)
- Same data = same datatype

User ✗

uid	Full name

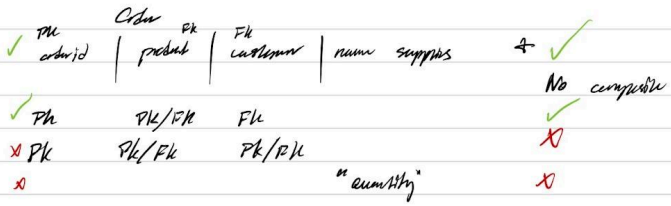
Full name is not atomic

1

2NF

- In 1NF
- Non-key attributes can't have partial dependencies

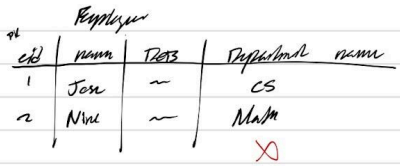
- Partial means its only reliant on part of the composite key



1

3NF

- 2NF
- The non-key attributes must not have transitive dependencies



Assume the table "department" exists
✗ Department name → did
Department name → eid

- Only happens when multiple tables

1

4NF ∞

Student

sid	Language	country of residence
1	english	USA
2	spanish	Spain
3	English	Spain
4	Spanish	USA

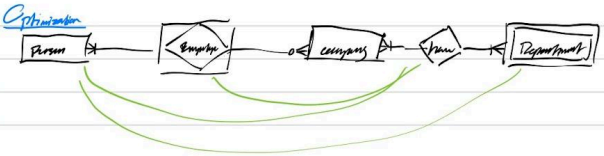
=>

- Avoid tables with multiple values

Student_Language	
id	Language

Student_residence	
id	residence

Optimization



- If company has only 1 department, then we can know if a person works at a department
- Create less amount of tables as possible

Milestone 1 Version 2

Executive Summary

Disasters and emergency situations cause distress as people are displaced from their homes, in need of shelter, food, and medical attention. Unfortunately, much crisis relief comes from outdated registries that have one-off guesses of what's needed and/or available. People cannot easily get what they need when they need it because there are no up-to-date sources of what's being offered and where, until now, with CrisisRelief. CrisisRelief seeks to serve as a portal for everything one needs to find temporary shelter, food banks, medical help, and natural disaster assistance, all in one place at the same time with real-time information. CrisisRelief fulfills the need for access in a timely fashion through utilizing APIs and hopes to include crowd sourced information in the future.

CrisisRelief utilizes the Google Maps API for geolocation, mapping, and emergency routing, to direct users to available resources. In addition, it also uses OpenWeatherMap API, for weather reporting and notifications that can warn users of bad weather that would hinder access to certain resources. What makes CrisisRelief unique is its live resource location and crisis direction. CrisisRelief has an ease-of-access interface for its web version so that those in distress don't have to parse through technology to receive help sooner and better.

Funding this project will enable the development for scalable, cloud-based infrastructure that ensures high availability and responsiveness in areas most needed. CrisisRelief also wants to enable community updates, allowing users and relief workers to report real time changes in things such as shelter capacity and food availability. CrisisRelief is set to become the default and only application required for any users in distress to find the help they need. Investing in CrisisRelief means investing in a smarter and more efficient way of providing aid to people in need of assistance.

Use Cases

Use Case 1: Finding Nearby Emergency Resources

Actors: John(Customer), CrisisRelief(My Company)

Assumptions:

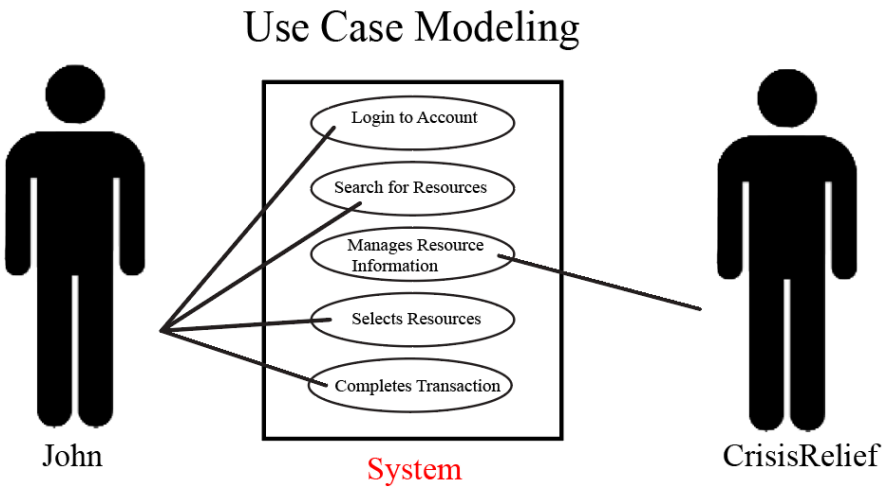
- John has internet access.
- John is in a stressful situation and needs quick help

Use Case:

John, who was affected by the first in LA, is looking for the nearest emergency shelter after his house has burned down. John decides to access CrisisRelief which helps display to him an interactive map of shelters, foodbanks, and medical centers that are nearby to him. After John selects a location, CrisisRelief will navigate John to that location in the quickest and safest way possible.

Benefits:

- Enables John to quickly locate emergency resources
- Provides clear navigation
- Gives real-time accessibility, helping to reduce anxiety.



Use Case 2: Real Time Updates

Actors: Mark (Relief Worker), General Public , CrisisRelief(My company)

Assumptions:

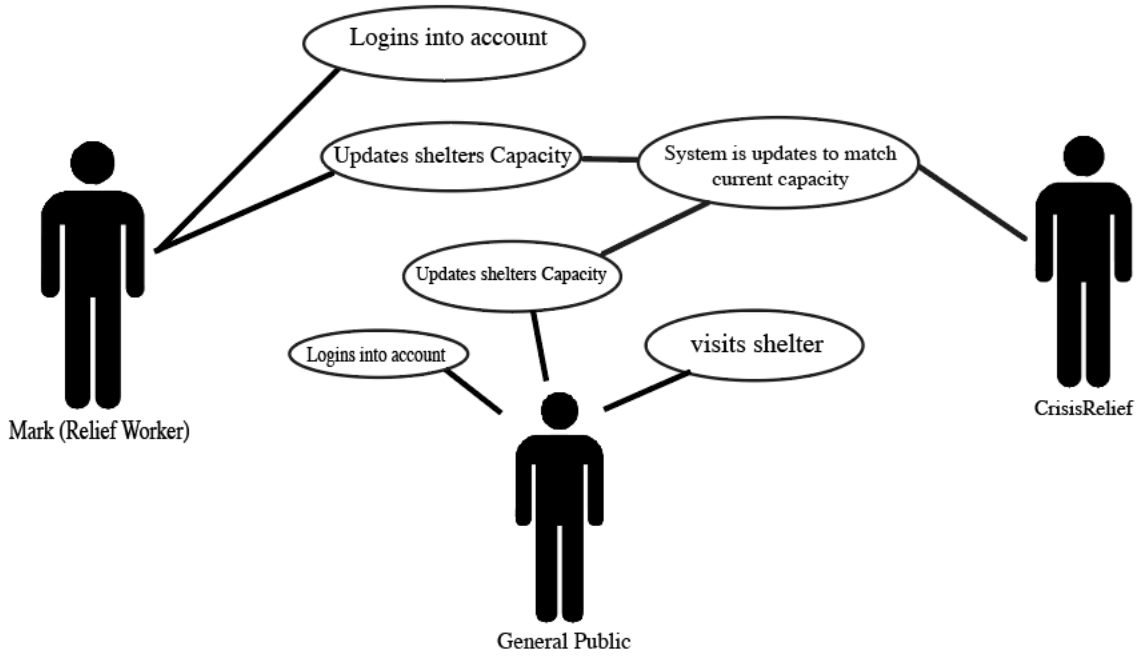
- Mark administrative access to update resource information
- Shelters and resource centers have internet access.

Use Case:

Mark, who is a relief worker at a local shelter, sees that the shelter capacity is almost at max. He goes in and logs into the system(CrisisRelief), and updates the capacity available from 60 down to 10. This real time update helps prevent miscommunication from word of mouth and also helps other people in need find available shelters. Mark also decides to mark that there is a water bottle shortage, which can prompt nearby relief locations to restock on supplies.

Benefits:

- Keeps emergency resource information accurate and up to date.
- Helps communication between shelters and the general public.



Use Case 3: Shelter Locator

Actors: James (Customer) CrisisRelief (My Company)

Assumptions:

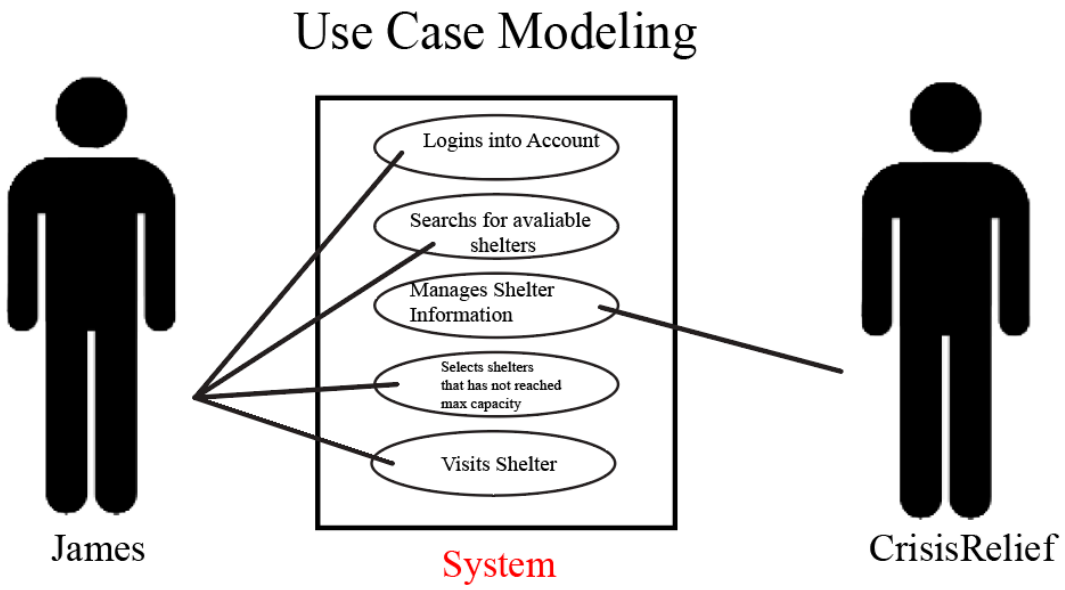
- James is houseless and currently does not have shelter
- Concerned about incoming weather emergency
- James does not have stable living condition

Use Case:

James has currently been evicted from his apartment and is now houseless. Due to cold winters in San Francisco, James wants a secure place to sleep so he logs onto CrisisRelief, which helps him locate nearby shelters with available space for the night. Once James arrives, he is able to find an open bed with blankets and secures his place for the night. James can now sleep in an available bed with blankets and update the website on availability.

Benefits:

- Access to immediate shelter and provide emotional relief
- James is now safe and reduces possible medical risks from not being in the storm.
- A sense of security from harsh weather conditions



Use Case 4: Medical Assistance

Actors: Lisa (An individual struggling with Diabetes) CrisisRelief (My Company)

Assumptions:

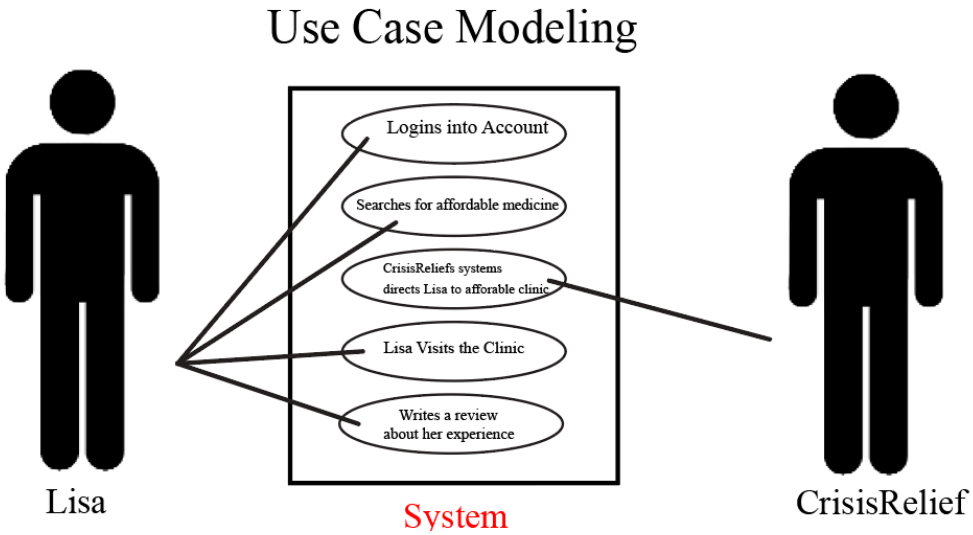
- Lisa has been diagnosed but does not have the finances to afford insulin
- Lacks a stable job and is struggling to find ways to pay for insulin and supplies
- Lisa has access to a mobile device and internet

Use Case:

Lisa, a 25-year-old struggling to manage her diabetes, often finds herself unable to afford insulin. Lately, she has noticed that buying one bottle of insulin requires her to compromise a week's worth of groceries. Lisa uses her smartphone and CrisisRelief, and is directed by the server side of CrisisRelief's system to a clinic that provides insulin at an affordable price. She goes to the location and buys the medicine she requires and submits a review about her experience at the clinic.

Benefits:

- Access to free medical care and immediate medical relief by a professional health provider.
- Gains a long term support system and reduce medical risks



Use Case 5: Real-Time Weather Warnings on Resource Pages

Actors: Kyle (Customer), CrisisRelief(My Company)

Assumptions:

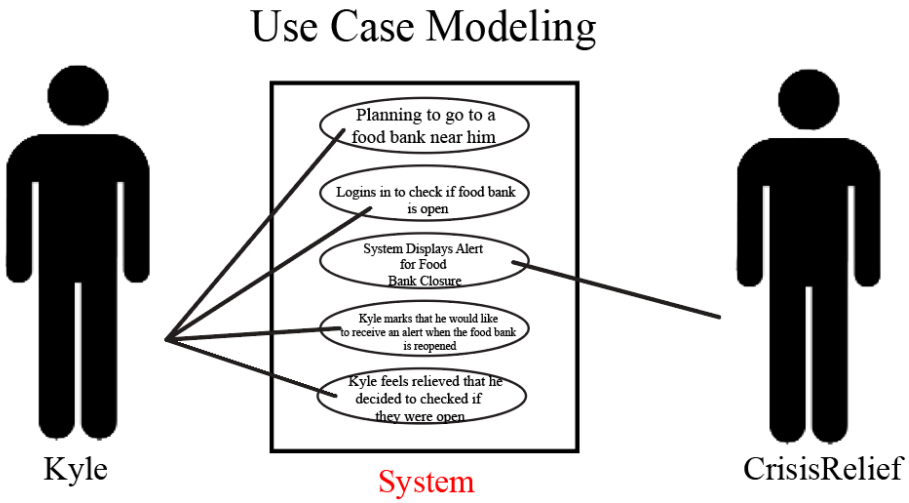
- Kyle has access to internet
- Weather conditions impact shelter accessibility and operations.

Use Case:

Kyle is planning to go to a food bank near him, but after taking a quick glance outside, he realizes that maybe the weather may not permit him to do so. He is concerned that if he was to go to the food bank, maybe it would be closed. Kyle decides to open a crisis relief platform(CrisisRelief), which allows him to search for the status of his food bank. After typing in his food bank, he sees that there is a warning symbol over the location. After clicking on it, Kyle realizes that his food bank is closed due to weather alerts. Kyle decides to mark that he would like to receive an alert when the food bank is reopened. Kyle now feels relieved that he decided to check whether or not the food bank was open, knowing that he would have risked his safety.

Benefits:

- Informs users about weather conditions that impact access to resources.
- Uses CrisisRelief servers directly for real-time weather alerts.
- Prevents unnecessary travel to closed or unsafe locations.



Use Case 6: Identifying High Risk Areas and Safe Zones

Actors: Samantha (Customer), CrisisRelief(My Company)

Assumptions:

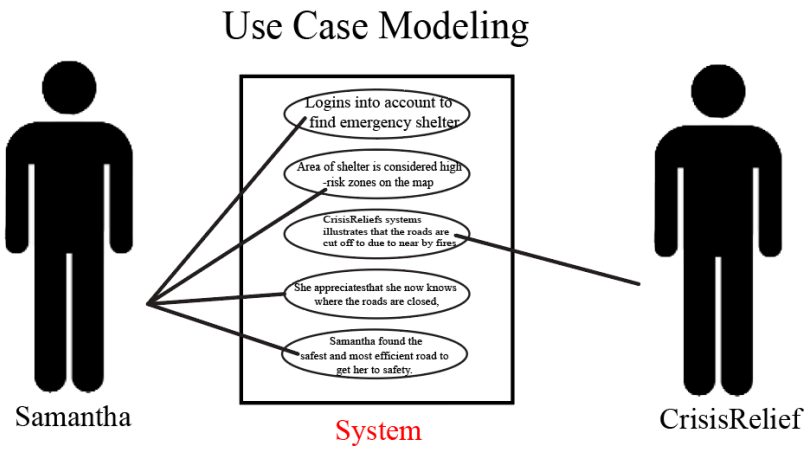
- Samantha has access to a smartphone with internet access
- Roads are closed due to hazards

Use Case:

Samantha needs to reach an emergency shelter, but due to the fires, she doesn't know what area is safe to go through. Samantha is concerned if the emergency shelter she is used to has roads cut off that would hinder her getting there. That is when she decides to use CrisisRelief. CrisisRelief shows Samantha areas that are considered high-risk zones on the map. She appreciates that she now knows where the roads are closed, and which is the safest way to get there. Reassured, she begins to drive, but gets a warning that another hazard is coming up, so she is redirected to a better road to get to her local shelter. By using real time traffic and hazard data, Samantha found the safest and most efficient road to get her to safety.

Benefits:

- Helps users avoid hazardous areas during an emergency.
- Provides real time updates on road closures and risks.
- Offers safer alternative routes that avoid hazards



Use Case 7: Mental Health Support Review

Actors: Jennie (Customer), CrisisRelief(My Company)

Assumptions:

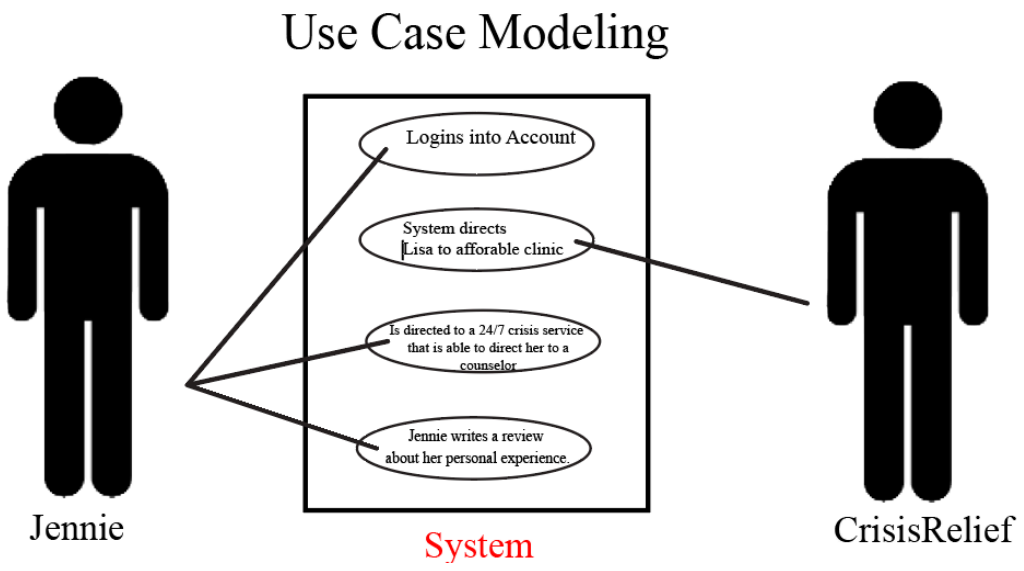
- Jennie has access to a smartphone or to the internet.
- Jennie feels overwhelmed and does not know where to seek help.
- Has no support system to turn to in a time of need.

Use Case:

Jennie has just been notified that she will be evicted from her current apartment by the end of week if she is unable to pay rent by the end of the week. Jennie is feeling stressed and anxious about the situation as she does not know if her upcoming pay will be enough to cover the cost of rent. She has not been able to sleep for the past few days and is really affecting her work abilities. Jennie searches on CrisisRelief and is directed to a 24/7 crisis service that is able to direct her to a counselor that provides Jennie the space to talk about her situation. Jennie is able to address her concerns and now has access to a counselor to provide her emotional relief. Jennie writes a review about her personal experience.

Benefits:

- Provides emotional support and creates a long term support system
- Free access to medical help and reduces future distress



Use Case 8: Food Bank Real-Time Update

Actors: Olivia (Customer), CrisisRelief(My Company),and James (food bank coordinator)

Assumptions:

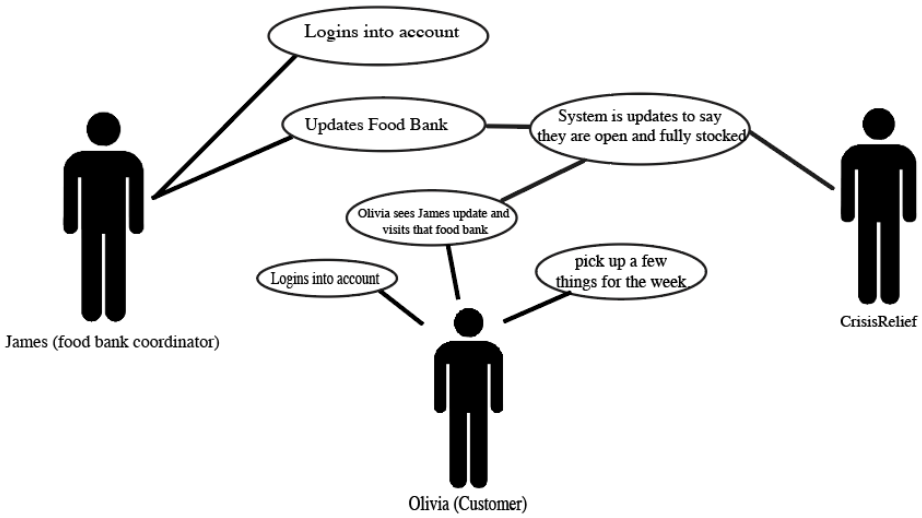
- Olivia had access to internet
- James is able to monitor the inventory in real time

Use Case:

Olivia is currently struggling to find a way to buy groceries for the upcoming week. She wants to visit a food bank to see if she is able to take any groceries home. She goes online and uses CrisisRelief to see if there are any food banks available near her that are open. James is currently working at the food bank and updates the website to state that not only are they open but full stocked as well. Olivia sees James update and is able to visit that food bank and pick up a few things for the week.

Benefits:

- Reduce food waste and people are able to get resources they need
- Real time tracking improves efficiency and allows more people to access resources faster



List of Main Data Items and Entities

1. Users

Description: Individuals who interact with the CrisisRelief platform.

Types of Users:

- **General Public**: Users in need of emergency assistance, food, medical help.
- **Relief Workers**: Verified personnel responsible for updating resource availability.
- **System Administrators**: Individuals with full access to monitor and manage the platform.

2. Emergency Resources

Description: Essential resources available for individuals affected by a crisis.

Types of Resources:

- **Shelters**: Locations providing temporary housing.
- **Food Banks**: Centers distributing free food to those in need.
- **Medical Facilities**: Clinics and hospitals offering medical aid.
- **Mental Health Support Centers**: Places providing psychological assistance.
- **Supply Stations**: Centers distributing emergency items (e.g., water, blankets).

3. Weather Alerts

Description: Real-time weather notifications affecting emergency resources.

Types of Alerts:

- **Severe Storms**
- **Hurricanes**
- **Flood Warnings**
- **Heatwaves**
- **Wildfires**

Functional Requirements

1. Functional Requirements for Resources

1.1. The system shall allow users to search for nearby emergency shelters.
1.2. The system shall display real-time availability of emergency shelters to users.
1.3. The system shall provide directions to the nearest emergency shelters for users.
1.4. The system shall notify users of real-time updates on shelter capacity.
1.5. The system shall show estimated travel times to shelters for users.
1.6. The system shall enable users to search for nearby food banks.
1.7. The system shall display real-time availability of food banks to users.
1.8. The system shall provide directions to the nearest food banks for users.
1.9. The system shall notify users of real-time updates on food bank inventory.
1.10. The system shall notify users of real-time updates on food bank restocking.
1.11. The system shall allow users to search for nearby medical centers.
1.12. The system shall display real-time availability of medical centers to users.
1.13. The system shall provide directions to the nearest medical centers for users.
1.14. The system shall notify users of real-time updates on medical center capacity.
1.15. The system shall show estimated travel times to medical centers for users.
1.16. The system shall provide access to mental health support services for users.
1.17. The system shall display high-risk areas on the map for users.
1.18. The system shall suggest alternative safe zones to users.
1.19. The system shall allow relief workers to update the capacity of emergency shelters.
1.20. The system shall allow relief workers to update the inventory of food banks.
1.21. The system shall enable relief workers to mark shortages of supplies.
1.22. The system shall allow relief workers to update the availability of medical centers.
1.23. The system shall enable relief workers to update the status of high-risk areas.
1.24. The system shall allow relief workers to update the status of alternative safe zones.

2. **Functional Requirements for Notifications & Alerts**
    2.1. The system shall alert users with real-time weather updates.
    2.2. The system shall notify users of real-time updates on shelter capacity.
    2.3. The system shall notify users of real-time updates on food bank inventory.
    2.4. The system shall notify users of real-time updates on food bank restocking.
    2.5. The system shall notify users of real-time updates on medical center capacity.
    2.6. The system shall notify users of critical updates on resource availability and conditions.
    2.7. The system shall allow users to set alerts for weather condition improvements.
    2.8. The system shall notify relief workers when users access their updates.
    2.9. The system shall allow relief workers to set up automated notifications for resource updates.
    2.10. The system shall enable system administrators to configure notification settings for users.
3. **Functional Requirements for User Features**
    3.1. The system shall allow users to view historical data on resource availability.
    3.2. The system shall provide users with the ability to save favorite locations.
    3.3. The system shall enable users to receive notifications on new shelters or food banks in their area.
    3.4. The system shall allow users to view user reviews and ratings of shelters and food banks.
    3.5. The system shall provide users with the ability to filter search results by resource type (e.g., shelter, food bank, medical center).
    3.6. The system shall allow users to customize notification preferences.
    3.7. The system shall provide a feedback mechanism for users to report issues or suggest improvements.
    3.8. The system shall allow users to view detailed information about each resource (e.g., address, contact information, operating hours).
    3.9. The system shall enable users to share resource information via social media or email.

4. **Functional Requirements for User Experience & Interaction**
    4.1. The system shall provide a search history feature for users.
    4.2. The system shall allow users to view user reviews and ratings of shelters and food banks.
    4.3. The system shall allow relief workers to view historical data on resource usage.
    4.4. The system shall enable relief workers to generate reports on resource availability and usage.
    4.5. The system shall allow system administrators to generate reports on system usage.
    4.6. The system shall enable system administrators to manage database backups.
    4.7. The system shall allow system administrators to monitor system performance
5. **User Authentication & Role Management**
    5.1. The system shall allow relief workers to log in.
    5.2. The system shall allow system administrators to manage user accounts.
    5.3. The system shall enable system administrators to configure user roles and permissions.
    5.4. The system shall allow relief workers to manage user contributions and reports.
    5.5. The system shall allow system administrators to manage user contributions and reports.
6. **Functional Requirements for Platform and Accessibility**
    6.1. The system shall provide multilingual support for relief workers.
    6.2. The system shall allow relief workers to access the platform via web browsers and mobile applications.
    6.3. The system shall provide accessibility features for users with disabilities.
    6.4. The system shall enable system administrators to manage system security.
    6.5. The system shall allow system administrators to update system settings.
    6.6. The system shall enable system administrators to manage API integrations.
    6.7. The system shall allow system administrators to monitor and manage API usage.
    6.8. The system shall allow system administrators to manage third-party integrations.
    6.9. The system shall allow system administrators to update the system with new features.
    6.10. The system shall enable system administrators to manage system maintenance schedules.

Non-Functional Requirements

1. Usability:
    1.1. The system shall have a simple and intuitive interface so users can quickly find emergency resources without confusion.
    1.2. Users shall be able to easily navigate through the system, even with limited tech skills
    1.3. Registered Users with no formal training shall be able to change the status of locations.
    1.4. Registered Users with no formal training shall be able to mark shortages.
    1.5. Users within poor environments, like tents, shall be able to complete their tasks.
2. System requirements:
    2.1. Users shall be able to complete their tasks through their smartphone, laptop, or desktop.
    2.2. CrisisRelief shall use Google Maps API to suggest locations of food banks, or emergency shelters.
    2.3. CrisisRelief shall use OpenWeatherMap API to gather information about the weather.
3. Performance requirements:
    3.1. Users shall be able to access the status of locations within 5 minutes.
    3.2. Registered Users shall be able to change the status of locations within 5 minutes.
    3.3. Weather information from CrisisRelief shall be in real time.
    3.4. Food bank information from CrisisRelief shall be in real time.
4. Storage
    4.1. CrisisRelief shall store data with MySQL.
    4.2. CrisisRelief servers shall be hosted with Amazon EC2.
    4.3. The database shall be relational.
    4.4. Datagrips shall be used to update the database.
5. Privacy
    5.1. The data of the user shall not be sold.
    5.2. The only data that CrisisRelief shall collect from the user is their name, what help they need, and where they are located.
    5.3. When a user logs in, they shall receive a session token that lasts for 5 hours.
6. Content
    6.1. The information given shall be easily read through text.
    6.2. Images of nearby locations shall be shown to the user if available.
    6.3. The directions to locations shall be easily readable.
7. Marketing
    7.1. CrisisRelief shall easily be found by people looking for CrisisRelief.
    7.2. CrisisRelief shall use SEO to make it easier to find on search engines like Google.
    7.3. CrisisRelief shall have social media pages to help visibility.

Competitive Analysis

Identify and Compare Features

Feature/ Company	2-1-1 Bay Area	FindHelp.org	San Francisco-Marin Food Bank	SF.gov	Weather.com
Strengths	Privacy, major emphasis on customer service and providing direct help	Very direct, brings information forward immediately, organized	Beautiful design, detailed means for donating and receiving, localization	Informational, official with SF	In-depth information for different needs, nicely customizable
Weaknesses	Menus and information are intimidating at a glance	Menu/search heavy, too broad of results	Location restricted, site heavily caters to donors despite an emphasis on finding food	Awkward navigation, surface level	Inconsistent or hidden feature placement
Pricing	N/A	N/A	Permits one-time and monthly donations	N/A	N/A
Social Media	N/A	Instagram, X, Facebook, Youtube, LinkedIn	On-site blog, Instagram, X, Facebook, YouTube	Facebook, X	Instagram, X, Facebook, Youtube
Onboarding Experience	Direct with phone calls but menu-heavy otherwise	Concise	Informational but requires some digging for services	Concise but harder to navigate	Simple, clear, and smooth

High-Level Comparison

Feature	2-1-1 Bay Area	FindHelp.org	San Francisco-Marin Food Bank	SF.gov	Weather.com	Crisis Relief
User Registration	-	++	+	-	++	++
Food Bank Information	+	+	++	+	-	++
Real-Time Availability at Food Banks	-	-	-	-	-	+
Immediate and Direct Weather Alerts	-	-	-	-	-	+
Specific and Guided Medical Resources	-	-	-	-	-	+

Summary of Advantages and Competitive Edge

Much of what is currently provided on the web in regard to crisis-related services either serves one specific niche or too broadly displays surface-level information sacrificing specifics and convenience. Where CrisisRelief excels and separates itself in this field is underlined with its one-stop-shop aspect refined to be keener on an individual's needs. User registration is typically restricted to a simple email newsletter or a personal database, however, CrisisRelief aims to help individuals quickly through live updates in a way they want to be updated. Rather than periodic site visits, skimming through lists, or cell phone services as seen in its competitors, CrisisRelief's real-time notifications bring help to its users. This user-first focus also opens the door to more active community-driven contributions, especially as a platform so present in one's day-to-day. CrisisRelief doesn't just improve quality of life, it protects it.

Checklist

- ☑ The team has found a time slot to meet outside of class.
- ☑ GitHub Master has been chosen.
- ☑ The team has collectively decided on and agreed to use the listed software tools and deployment server.
- ☑ The team is ready to use the chosen front-end and back-end frameworks, and those who need to learn are actively working on it.
- ☑ The Team Lead has ensured that all members have read and understand the final M1 before submission.
- ☑ GitHub is organized as discussed in class (e.g., master branch, development branch, folder for milestone documents, etc.).

High Level System Architecture and Technology Used

- Server: AWS EC2
- Operating System: Amazon Linux 2023
- Database: PostgreSQL 16
- Backend Language: TypeScript
- Frontend Language: TypeScript

Additional Technologies

- Google Maps API
- OpenWeatherMap API (Weather Alerts)
- Frontend Framework: React
- Backend Framework: Node.js
- IDE: Visual Studio Code
- SSL Certificate: Let’s Encrypt (Cert Bot)
- Docker: Docker Container on the EC2 instance
- Database Design Framework: DataGrip 2024.3.4
- Redis
- WebSockets
- DuckDNS

List of Team Contributions:(Team Lead)

Kyle Nguyen (Documentation Lead) - 10/10

- Led the documentation efforts, ensuring the structure and content were well-organized.
- Assigned sections to team members and oversaw the completion of Milestone 1 Documentation.
- Authored the Use Case section, ensuring clarity and completeness.
- Attended all team meetings and actively participated in discussions.
- Incorporated all feedback provided and made necessary corrections to improve the documentation.

Karla Cardenas Andrade (Database Management Lead) - 10/10

- Took charge of database design and management.
- Contributed to use case development and created diagrams that enhanced clarity in documentation.
- Attended every meeting and took suggestions positively, building upon them.
- Provided guidance on database implementation and approved the final database setup.

Geoart Corral (GitHub Master) - 10/10

- Managed GitHub branches, ensuring smooth version control and collaboration.
- Regularly updated the README file and maintained project structure.
- Authored the Competitive Analysis section in the documentation, which was thoroughly detailed and well-researched.

Ayesha Irum (Frontend Lead) - 10/10

- Led the frontend development efforts.
- Completed the Functional Requirements section with precision.
- Designed the UI for the “About Us” pages.
- Proposed the idea to separate the “About Us” page into two:
- One page listing all members.
- A second page with individual team member details.

Francis Aviles (Backend Lead) - 7.5/10

- Led the backend development, focusing on core functionality.
- Authored the Non-Functional Requirements section.
- Attended all team meetings and actively participated in discussions.
- Incorporated all feedback provided and made necessary corrections to improve the documentation.
- Tried to set up the Database but was not able to complete it under the internal team deadline.
- No database was set up. For Milestone 1.