Cover Page

Document Title : Final Report
Document Purpose : A combination of all previous documents combined into one to showcase the evolution of the project from start to finish.
Project Title : Crowdsourced Disaster Relief Platform
Class : CS3354.004 , Feng
Group : 2
Members : Casey Nguyen, Kevin Pulikkottil, Andy Jih

Table of Contents :

Individual Contributions :

Responsibility Matrix:

- Responsible
- Accountable
- Consulted
- Informed

Activity	Casey	Kevin	Andy
Project Proposal	R A	R	R
Project Requirement Doc.	R A	R	R
1_code	R A	R	R
2_data_collection	R A	R	R
3_basic_function_testing	R C	R A	I
4_documentaion	R A	R	I
Management	R A	I	I
Planning	C I	R A	I
Frontend	R A	I	R
Backend	A C	R	I
Artificial Intelligence	I	R A	I
User Interface	R A	I	R
Final Report	R A	R	R

Responsibility Allocation Chart:

Activity	Casey	Kevin	Andy
Project Proposal	✅	✅	✅
Project Requirement Doc.	✅	✅	✅
1_code	✅	✅	✅
2_data_collection	✅	✅	✅
3_basic_function_testing		✅	
4_documentaion	✅	✅	
Management	✅		
Planning		✅	
Frontend	✅		✅
Backend	✅	✅	
Artificial Intelligence		✅	
User Interface	✅		✅
Final Report	✅	✅	✅

Work Assignment:

Sub-Teams:

1. Casey and Kevin
2. Casey and Andy

Individual Competences:

Casey:

- Frontend/Backend Developer, Management, Communication
- Worked on UI of Website, Oversaw the front and backend

Kevin:

- Backend Developer, Planner, Communication
- Worked on AI matching, Database storage. Did most the backend

Andy:

- Frontend Developer
- Worked on UI of Website

Project Management:

Product Ownership and Responsibility Breakdown:

Casey Nguyen: Oversee project management; contribute to both front and back ends. Does the pdfs and write ups. Oversees submission and final calls.

Kevin Pulikkottil: Lead backend development and AI matching implementation.

Andy Jih: Assist in front-end development.

Tools Used:

Github - For version control and to oversee each others code

Discord ./ Messages - For communication

VSCode - IDE

Flutter - For 1_code and the main website builder

Google Doc - For pdf write ups

Meetings:

Would text each other when we would work on projects and when due dates were. Would just work on it when we had time. Mostly through text and discord. Most of the time was just progress checks and what we needed to do before the due date.

Development Timeline:

Feb 16: Submitted Project Proposal

March 9: Submitted Project Requirement Documentation

March 10-16: Discussed next steps, what IDE we would use, what programming we would use (Flutter), and confirmed our project key features and how we wanted it to streamline data with AI

March 17-23: Spring Break

March 24-30: Launched respective front end and back end parts. To be integrated in Deliverable 2

March 31 - April 6: Finished basic UI for deliverable 1, finished key functions of the website, finished APIs and most of backend main points

April 7 - 11: Finished backend AI Matching, finalized all parts for submission. Finished respective pdfs.

April 11: Submitted all documents and code to eLearning for Deliverable 1

April 14 - 20 : Worked on remaining tweaks and features. Enhanced UI of website and integrated complete website

April 21 - 27 : Finished up website for presentation demo. Finished Final Product and Presentation.

April 28 : Presentation

April 29 - May 2 : Complete Website launch

May 4 : Submitted Project Deliverable 2

Summary of Changes :

- Vastly Improved UI from Deliverable 1. The UI from Deliverable 1 was basic as we were focused on the main functions. So we knew our website had to drastically evolve. This was evidenced by both submissions.
- Cut down to 5 main features: AI Matching, Resource Inventory, Emergency ALerts, Request Help, Donate. This was done primarily to focus and implement core features in a timely manner. These were the most important features so we cut out all the other ones to focus on this. Time, Cost and functionality played a big role in this decision.
- Front and Backend completely integrated since Deliverable 1. We didn;t fully integrate due to teammate issues where we had a member not due to his work. So we were behind. But we picked up the slack and got it done in a timely manner.
- Database correctly stores all data inputted by user in all screen and functions
- Sign Up / Sign In is ready for use. Again this was implemented after dealing with said teammate.
- Added Stripe checkout for donation handling of credit card payments. Originally we had a simple form where users can input x amount of money and just hit donate. This was basic and lazy so we needed to add a real payment processing system. We got the idea from our nice TA and we implemented that the very next day using Stripe Checkout. This was easy to integrate. Stripe is well known as well so people who are hesitant can feel a bit better.
- Added mobile app sizing and improvements. For deliverable 1, we originally had everything coded as a website for someone using on a laptop. Portability was one of the main goals since the proposal. So we knew we had to make it look better as if someone were to use the website on the phone. This was done by sizing down the website when someone is on mobile.

Customer Statement of Requirements :

Real-Time Communication and Reporting

- Ensure that disaster victims can report their needs quickly and accurately.
- Provide a way for volunteers, NGOs, and first responders to access and share real-time updates.
- Maintain communication channels even in areas with poor connectivity.

Resource Allocation and Distribution

- Develop a system to match available resources (food, water, medical aid) with the most urgent needs.
- Optimize resource distribution based on proximity and priority.
- Prevent duplicate or misallocated relief efforts by different organizations.

User Identity and Role Management

- Implement a secure and safe registration system to verify victims, volunteers, and NGOs.
- Prevent unauthorized users from accessing sensitive information or manipulating resources.
- Assign role-based access controls to ensure different stakeholders have the appropriate permissions.

Scalability and System Reliability

- Ensure the platform can handle a large number of users during peak disaster situations.
- Implement cloud computing and load balancing for system time and performance.
- Design offline functionality to allow continued use in areas with unstable networks.

Data Security and Privacy

- Securely handle sensitive user data, including personal and financial information of donors.
- Implement encryption and authentication features to protect against unauthorized access.

- Address potential risks related to data breaches and fraudulent activities.

Trust and Safety Measures

- Develop features to verify the legitimacy of NGOs and volunteers before they access critical resources.
- Implement safeguards to prevent misinformation or fraudulent activities within the platform.
- Ensure accountability and transparency in disaster response efforts.

Glossary of Terms :

1. **Disaster Victims** - Individuals directly affected by a disaster who require assistance such as shelter, food, or medical aid.
2. **Non-Governmental Organizations (NGOs)** - Independent organizations that provide disaster relief, including food distribution, medical aid, and shelter assistance.
3. **Relief Organizations** - Entities, including NGOs and government agencies, that coordinate disaster response and distribute aid to affected areas.
4. **Real-Time Reporting** - A system feature allowing users to provide and access live updates on disaster situations, including resource availability and urgent needs.
5. **Resource Matching** - The process of connecting available resources (e.g., food, medical supplies, shelters) with affected individuals or areas based on urgency and proximity.
6. **Role-Based Access Control (RBAC)** - A security mechanism that assigns different levels of access to users based on their roles (e.g., victims, volunteers, NGOs).
7. **Crowdsourced Data** - Information collected from a large number of users, including victims and volunteers, to create a complete overview of the disaster situation.
8. **Offline Functionality** - A feature enabling users to access critical information and report needs even when internet connectivity is limited or unavailable.
9. **Load Balancing** - A method to distribute user requests across multiple servers to ensure system stability and prevent overload during peak disaster situations.
10. **Two-Factor Authentication (2FA)** - A security feature requiring users to verify their identity using two independent methods to enhance data protection.
11. **Misinformation Control** - Measures taken to prevent the spread of false or misleading information within the platform to ensure accurate disaster response.
12. **Donors** - Individuals or organizations providing financial or material aid to support disaster relief efforts.
13. **AI Matching** - refers to the use of artificial intelligence techniques (such as machine learning, natural language processing, or deep learning) to automatically pair or align items, people, or data based on certain criteria, preferences, or patterns.

System Requirements :

The Crowdsourced Disaster Relief Platform backend must satisfy the following requirements:

1. Real-Time Help Requests
    - Victims submit requests (type, description, geolocation) via the API.
    - The system records requests in Firestore and pushes them to subscribed clients or queues them offline.
2. Live Updates & Emergency Alerts
    - Authorized users (Admins/NGO staff) post alerts (message, timestamp) via Firebase.
    - All clients receive updates instantly through Firestore's real-time listeners.
3. AI-Driven Matching
    - On request creation, a MatchingService retrieves all volunteers' firebaseUid and locations.
    - It computes Haversine distances and assigns the nearest available volunteer by updating the request's assignedVolunteerId.
    - If no volunteers are available, the request remains unassigned for manual pickup.
4. Resource Inventory Management
    - Volunteers/NGOs add resources (type, quantity, location).
    - Clients query, filter, and sort resources (by stock level or region) via Firestore queries.
    - New entries propagate automatically to all viewers.
5. Donation Processing
    - Donors initiate a Stripe Checkout session from the backend (/donate endpoint).On success, a Donation document is created (amount, donor Aid, timestamp).
    - Backend handles Stripe webhooks to confirm transactions, ensuring PCI compliance.
6. Secure Authentication & Roles
    - Firebase Email/Password Authentication is fully configured (Auth provider enabled).
    - Users register/sign in via FirebaseAuth.createUserWithEmailAndPassword() and signInWithEmailAndPassword().

    - Each User has a role (Victim, Volunteer, NGO, Admin, Donor) and firebaseUid.
    - Firestore security rules enforce RBAC: only allowed roles can write to Alerts, Resources, or Requests.
7. Data Security & Privacy
    - All read/write operations occur over HTTPS. Service account keys are secured; sensitive fields excluded via .gitignore.
    - Donations handled via Stripe-no card data stored on our servers.
8. Scalability & Fault Tolerance
    - Hosted on Google Cloud (Firestore auto-scales).
    - Stateless backend services (FastAPI/Unicorn) can be horizontally scaled behind a load balancer.
    - Firestore's offline persistence ensures brief connectivity outages do not block critical operations.
9. Audit & Transparency
    - Every action (create/update of Requests, Alerts, Resources, Donations) logs a timestamp and user ID.
    - These logs allow post-incident analysis and traceability.

Functional Requirements Specification :

Implemented Use Cases

1. Submit Emergency Help Request
    - Actor: Victim
      Flow: Victim fills out a form; the client calls POST /requests. Firestore stores a Request doc. A Cloud Function (or backend listener) triggers MatchingService.findVolunteerForRequest(requestId), which assigns a volunteer and updates the document.
    - Success: Request appears in the volunteer's dashboard and on the map.
    - Alternate: If offline, the client queues the submission and syncs upon reconnection.
2. User Registration & Authentication
    - Actors: New User, System
    - Flow: User signs up with email/password. Client invokes FirebaseAuth.createUserWithEmailAndPassword(). On success, a User doc (with role and firebaseUid) is created in Firestore. Login uses signInWithEmailAndPassword(), returning an ID token for authenticated API calls.
    - Error Handling: Invalid credentials yield UI errors; Firestore rules block unauthorized writes. Password reset links can be sent via Firebase console.
3. Provide Emergency Alerts
    - Actors: Admin/NGO, All Users
    - Flow: Admin calls POST /alerts; Firestore writes an Alert doc. All clients subscribed to the alerts collection immediately display the new entry.
4. Manage Resource Inventory
    - Actors: Volunteer/NGO, Any User
    - Flow (Add): Authorized user calls POST /resources. Firestore adds a Resource doc linked to providerUid.
    - Flow (View/Search): Users query /resources?filter=&sort=; backend returns matching docs.

5. Make a Donation
    - Actors: Donor

    - Flow: Donor requests /donate?amount=X. Backend creates a Stripe Checkout session and redirects. On webhook confirmation, backend writes a Donation doc with donor's firebaseUid, amount, and timestamp.

Deferred Future Use Cases

- Volunteer Progress Tracking
- Offline Request Handling
- Content Moderation & Verification
- Advanced AI Resource Optimization

A full Traceability Matrix links each use case to the system requirements above.

Effort Estimation :

Home Screen: Effortless. Able to view all functions of the website right when you get on the website. Can scroll down to read user stories and to read how the basis of our website works.
Resource Inventory: One click from homescreen. Able to see available resources right then and there. Able to click filters and sorts to see resources that are low and by city. Able to then easily see the add resource button to add a resource onto the website.
Emergency Alerts: One click from the homescreen. Able to scroll up and down to see current emergencies in the areas.
Request Help: One click from homescreen. Able to easily access the form to fill out for help. Getting location is easy as our geo-locator pinpoints your exit location with 2 clicks. You are then able to scroll and see a current list for help. A map is also provided for visual representation of the requests for help.
Donation: One click from the homescreen. Form is pinned to the top for easy use. Hitting donate takes you to the stripe checkout where it guides you easily to fill out your information. From the donation screen you are able to see previous donations with scrolling
Sign Up / Sign In: One click from the homescreen. Simple and basic form to fill out email and password to get in.
Navigation: Most things to do on the website are 1-3 clicks. Very easy and straightforward.

Domain Analysis :

We identified these primary domain concepts and their mappings:

Domain Concept	Description & Mapping
User (Victim, Volunteer, NGO, Donor, Admin)	Unified into a single User class with attributes: firebaseUid, name, role, location (lat/long). Auth via Firebase Auth.
Request	Represents a help request. Fields: requestID, description, location, creatorUid, assignedVolunteerId, status.
Resource	Inventory entry. Fields: resourceID, type, quantity, location, providerUid.
Alert	Emergency notification. Fields: alertID, message, timestamp, createdByUid.
Donation	Financial contribution. Fields: donationID, amount, donor Aid, timestamp.
AI Matching	Service module (MatchingService) mapping Requests → Volunteers based on proximity.

Key Relationships:

- User-Request: Victim creates Request; Volunteer is assigned to Request.
- User-Resource: Volunteer/NGO (provider) adds Resource; any User views it.
- User-Alert: Admin/NGO (creator) posts Alert; all Users read it.
- User-Donation: Donor creates Donation; platform records it.

This domain model ensures each use case is underpinned by corresponding entities and associations. A concept-to-class traceability matrix confirms full coverage: every requirement and use case is grounded in these domain constructs.

Class Diagram :

In the design phase, we transformed the domain model into a set of software classes, data structures, and interfaces that make up the backend of the Crowdsourced Disaster Relief Platform. The Class Diagram below shows the key classes in our system and their relationships. We then describe each class/interface and how the domain concepts evolved into these classes, including details of important methods (interface specifications) and how the AI matching logic is realized in the design.

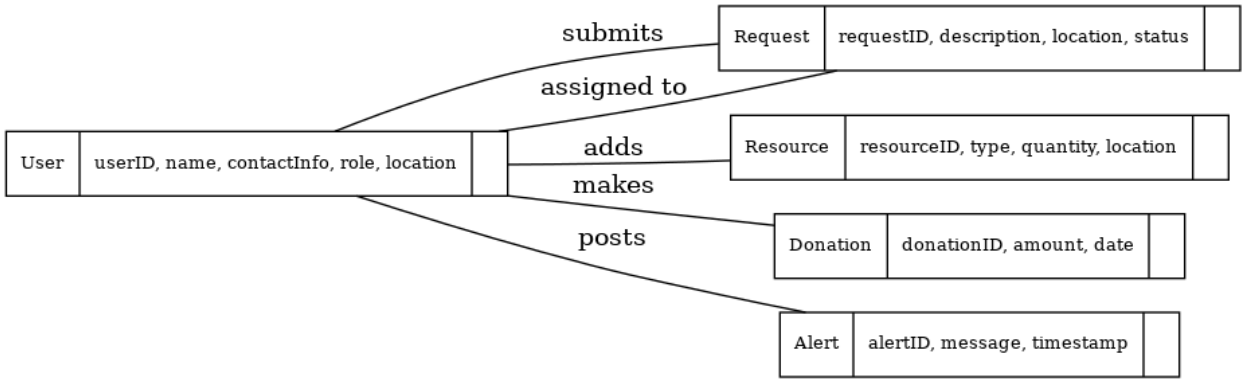


Figure: UML Class Diagram illustrating the key backend classes and their relationships in the Crowdsourced Disaster Relief Platform. The design consolidates various user roles into a single User class (with a role attribute) connected to core entity classes: Request, Resource, Donation, and Alert. Each relationship reflects how data flows between users and these entities (e.g., a User submits a Request, a Request is assigned to a User as a responder, a User adds a Resource, makes a Donation, or posts an Alert). This class structure is implemented in the Firebase data schema and through Flutter/Dart classes in the application.

Overview of Design Classes:

- User
    - Unified actor for Victims, Volunteers, NGOs, Donors, Admins
    - Key fields: firebaseUid, name, role, location
    - Interfaces to register/login via Firebase Auth and enforce role-based access
- Request

    - Represents a help request: requestID, description, location, creatorUid, assignedVolunteerId, status
    - Methods to create a request and assign a volunteer
- Resource
    - Inventory item: resourceID, type, quantity, location, providerUid
    - Supports adding and updating resource entries
- Donation
    - Financial contribution: donationID, amount, donor Aid, timestamp
    - Recorded via Stripe webhooks into Firestore
- Alert
    - Emergency broadcast: alertID, message, timestamp, createdByUid
    - Published to all clients via Firestore's real-time updates
- MatchingService
    - Core AI module that matches a Request to the nearest available Volunteer
    - Computes distances and updates assignedVolunteerId on the request

Concepts-to-Classes Evolution

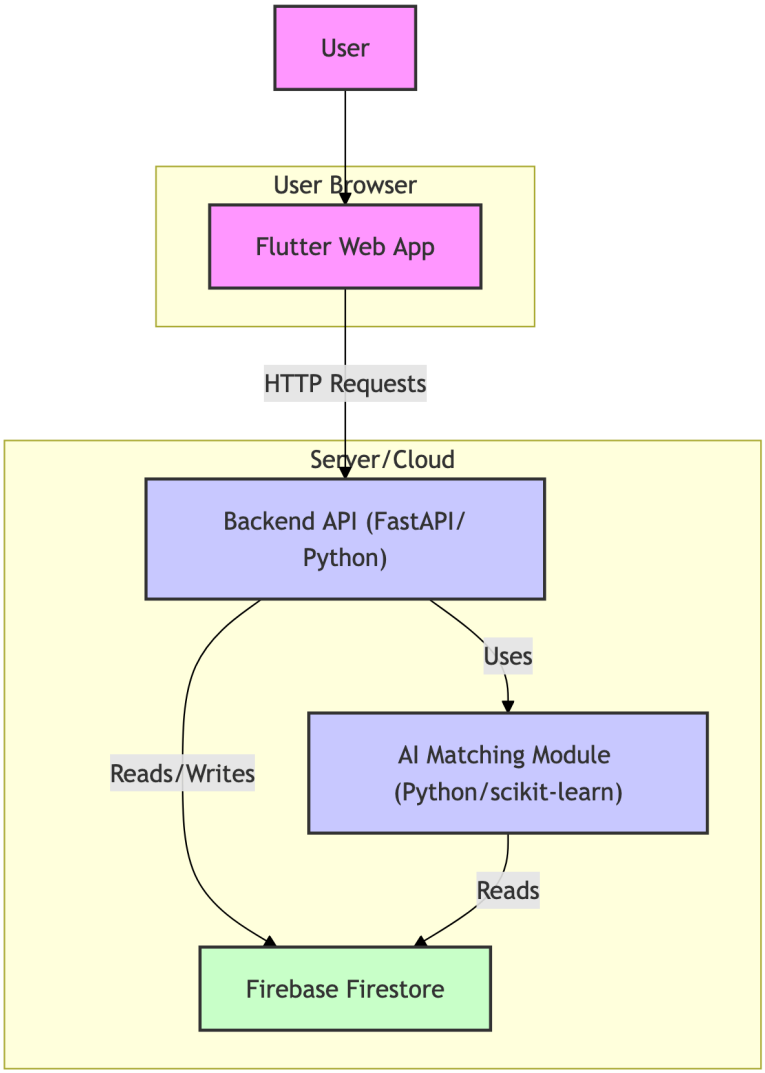
- Victim / Volunteer / NGO / Donor / Admin → User
  All actor roles unified into one User class. Distinguished by a role field rather than separate classes. Common fields (firebaseUid, name, location) and shared methods (register(), login(), isAuthorized()).
- Help Request → Request
  Domain Request maps to Request class with requestID, description, location, creatorUid, assignedVolunteerId, and status. Methods for creation and volunteer assignment encapsulate the match workflow.
- Resource → Resource
  Each inventory item becomes a Resource class: resourceID, type, quantity, location, providerUid. Supports simple CRUD (add/update) aligned with the domain concept.
- Donation → Donation
  Financial contributions map directly to the Donation class (donationID, amount, donor Aid, timestamp), created via Stripe webhook handlers.
- Emergency Alert → Alert
  Alerts become the Alert class (alertID, message, timestamp, createdByUid) and published via Firestore's real-time listeners.
- AI Matching → MatchingService
  The abstract matching process is implemented as a MatchingService module. It

retrieves users with role = Volunteer, computes distances, and updates requests with the nearest volunteer.

- Additional Domain Concepts
    - User Verification / Misinformation Control → reflected as user attributes (e.g. isVerified) and security rules, not separate classes.
    - Location → treated as a shared attribute rather than its own class.

System Architecture and System Design :

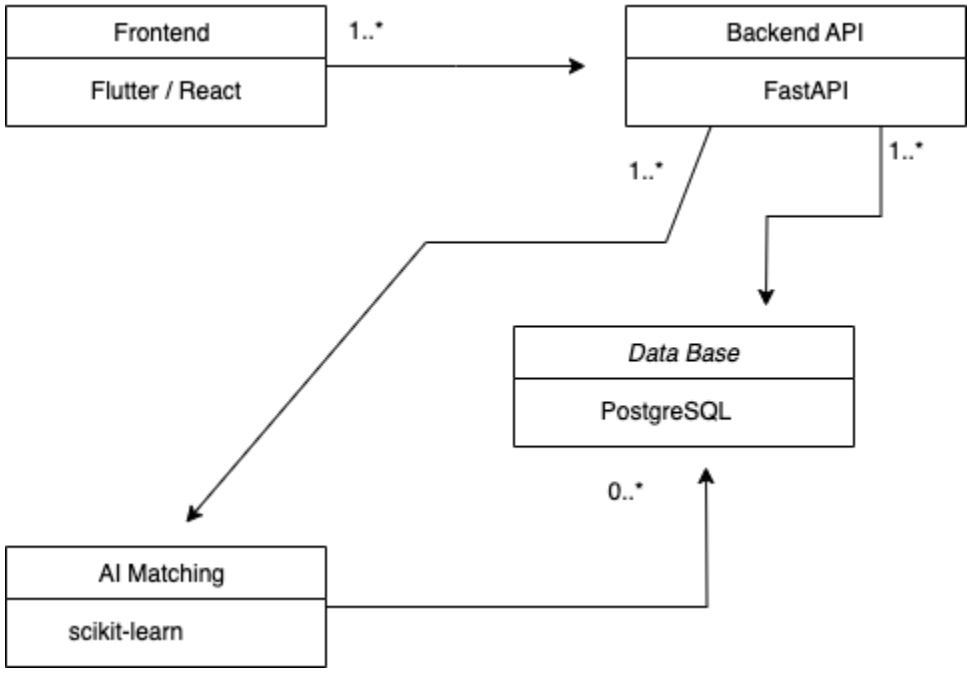
High Level Diagram:



UML Diagram :

Diagram Explanation:

- The Frontend Subsystem communicates with the Backend API via HTTP requests.
- The Backend API Subsystem interacts with the Database Subsystem using SQLAlchemy ORM (JDBC).
- The AI Matching Subsystem receives data from the Backend API, processes matches, and returns results.



Overview:

The Crowdsourced Disaster Relief Platform is designed as a modular web system that supports real-time disaster response through data-driven matching, user input, and secure data handling. It consists of a Flutter-based frontend, a FastAPI backend, and a Firebase Firestore database. Optional support for Dockerized deployment is also included.

Technologies Used:

- Backend: Python, FastAPI, Firebase Firestore, scikit-learn, NumPy, Geopy, Uvicorn, Joblib, Requests. Docker support is optionally available.
    - A RESTful API using FastAPI
    - Interfaces directly with Firebase Firestore to persist and retrieve data
    - Provides 3 key endpoints:
        - / - health check root
        - /match/{request_id} - production AI matching endpoint
        - /debug-match/{request_id} - verbose KNN diagnostics and volunteer info
    - Incorporates a modular AI logic module (matching_ai.py) that handles location geocoding, skill encoding, and distance-based KNN volunteer matching
- Frontend: Flutter, Dart.
    - Built using Flutter and written in Dart
    - Provides a multi-platform UI accessible via web and mobile devices
    - Communicates with the backend using HTTP requests (e.g., matching, data submission)
    - Implements navigation through reusable screens for:
        - Resource Inventory
        - Emergency Alerts
        - Help Requests
        - Donations (with Stripe Checkout)
        - User Sign-up/Sign-in
- Database: Firebase Firestore.
    - NoSQL cloud database used to store:
        - Volunteer info (location, skills, availability)
        - Aid requests (location, urgency, assistance type)
        - Alerts, resources, and donations

- AI Matching Engine:
    - Located in matching_ai.py
    - Extracts features from both requests and volunteers using:
        - One-hot encoding (for type/skills)
        - Geolocation via GeoPy
        - Urgency or availability encoded numerically
    - Computes matches using K-Nearest Neighbors (KNN) with Euclidean distance
    - Debug endpoint provides detailed matching metadata including feature vectors and similarity distances
- Deployment Architecture
    - Docker is supported for backend development and deployment:
        - docker-compose.yml builds the API container
        - Firebase credentials are mounted securely into the container
    - Frontend is developed in Flutter and launched via flutter run
    - Project structure includes modular directories: 1_code/ (Flutter + backend), 2_data_collection/, 3_basic_function_testing/, and 4_documentation/
- Development Environment: VSCode.
- Testing: Pytest.

Design Decisions

- Frontend and backend are developed independently, allowing for flexible iteration and future scaling.
- Firestore was chosen over SQL due to its document-based schema, which is better suited to storing dynamic user-driven content like disaster requests and volunteer profiles.
- Given the size and scope of the project, KNN was selected as the AI model for its simplicity, interpretability, and real-time matching performance.

- Flutter enabled us to design responsive screens across web and mobile devices with a single codebase.

Algorithms and Data Structures :

AI Matching Algorithm

A K-Nearest Neighbors (KNN) algorithm is implemented using scikit-learn. This model automatically matches volunteers to aid requests based on a combination of geographic proximity, skills, and request urgency.

- Feature Extraction:
    - Aid requests include: type, location, and urgency.
    - Volunteers include: skills, location, and availability.
    - Locations are converted to latitude/longitude using the Geopy geocoding API.
    - Skills and types are one-hot encoded using OneHotEncoder.
    - Features are then scaled using StandardScaler to normalize distances.
- Matching Pipeline (matching_ai.py)
    - Construct feature vectors for both requests and volunteers.
    - Use NearestNeighbors from sklearn.neighbors to calculate Euclidean distances.
    - Return the top 3 closest volunteers for each request.
- Debug Endpoint (/debug-match) provides internal vectors, distances, and matching rationale.

Backend Data Handling

The backend runs a FastAPI application that communicates with Firebase Firestore, a NoSQL cloud database. Core collections include:

- requests: Contains active help requests.
- volunteers: Lists available volunteers with their metadata.
- donations, resources, and alerts: Store respective domain-specific entries.

Data is represented and exchanged in JSON format, aligned with the frontend Dart models.

Frontend Data Models

We use Dart models to structure data across various screens, each aligned with corresponding backend collections.

- User

```
class User {
```

```
  final String email;
  final String password;
}
```

- Request

```
class Request {
  final String name;
  final String type;
  final String description;
  final double latitude;
  final double longitude;
}
```

- Resource

```
class Resource {
  final String name;
  final int quantity;
  final String location;
}
```

- Donation

```
class Donation {
  final String name;
  final String type;
  final String detail;
}
```

- Alert

```
class Alert {
  final String alertTitle;
  final String alertDescription;
  final String alertLocation;
  final String alertDate;
}
```

Each model includes a fromJson() and toJson() method for serialization and integration with Firestore. This structure promotes modularity and type safety.

User Interface Design and Implementation :

The user interface (UI) of the Crowdsourced Disaster Relief Platform was designed with simplicity, responsiveness, and usability in mind. The frontend is built entirely using Flutter for the web, allowing for fast development and an inherently responsive design.

Overall Structure:

- HomeScreen (home_screen.dart) serves as the starting point of the application. Upon launch (main.dart), users are directed to a welcoming screen that offers access to five major features via animated buttons.
- Navigation is handled through Flutter's built-in Navigator.push() system. Each major feature (e.g., Request Help, Donate, Emergency Alerts) is accessible in one click from the HomeScreen.

Main Screens and Features:

- Home Screen
    - Features a logo, navigation buttons, and introduction sections like "User Stories", "Contact", "Team", and "About AI Matching."
- Request Help Screen (request_posting_screen.dart)
    - Allows users to submit a request for aid by entering their name, aid type (Medical, Food, Shelter), and a description. Users can capture their live geolocation using a built-in map integration (FlutterMap + Geolocator). Current requests are displayed in a list and on a map.
- Resource Inventory Screen (resource_inventory_screen.dart)
    - Displays a list of available resources with sorting (by city or quantity), filtering (show low inventory), and search functionality. Admins/users can add new resources via a pop-up form.
- Emergency Alerts Screen (emergency_alerts_screen.dart)
    - Displays a scrollable list of active and past disaster alerts. Alerts are visually coded with different icons and colors based on disaster type (e.g., flood, earthquake, wildfire).
- Donation Screen (donation_screen.dart)
    - Users can donate money (redirected to Stripe Checkout) or submit resource donations. A list of recent donations is displayed below the donation form.
- Profile Screen (Sign Up/Sign In) (profile_screen.dart)

    - Provides account management through sign-up and sign-in forms, including basic validation (email format, password strength). Integrates with Firebase authentication services for user management.
- Custom Navigation Bar (nav_bar.dart)
    - A simple navigation bar featuring the website's logo on the left and a Sign Up / Sign In button on the right. Button animations on hover add polish for desktop users.

Mobile Responsiveness:

- UI elements adjust dynamically based on screen size (Wrap, Flex, Expanded widgets).
- Buttons, forms, and cards are sized appropriately to be fully accessible on smaller devices like phones and tablets.
- Navigation and form elements remain highly usable even when resized to narrow screen widths.

Key Improvements:

- A more polished UI with custom gradients, animations, and mobile-friendly layouts.
- Stronger visual hierarchy: Clear headers, section breaks, and card-based layouts.
- Dynamic components: New features like map-based request visualization and dynamic filters for resource inventory.
- Stripe integration: Secure external flow for monetary donations via Stripe Checkout.
- Error handling: User-friendly error messages and loading indicators.

Design of Tests :

To ensure correctness, security, and reliability, we designed tests at multiple levels:

1. Unit Tests
    - Matching Algorithm:
        - Provide mock lists of volunteers with known coordinates and a test request.
        - Assert findVolunteerForRequest() returns the correct (nearest) volunteer or null if none available.
        - Test edge cases: equidistant volunteers, empty volunteer list.
2. Integration Tests
    - Database Flows:
        - Create a Request via API; verify Firestore document fields.
        - Simulate matching trigger; verify assignedVolunteerId is set.
        - Add a Resource and perform a query; assert returned results match filters.
    - Stripe Donation:
        - Use Stripe's test mode and card tokens to simulate successful and declined payments.
        - Verify that successful charges create a Donation document; declines produce no record.
3. Authentication Tests
    - Sign-Up/Login Flows:
        - Valid signup → returns a Firebase token and creates a User doc with correct role.
        - Invalid signup (weak password, existing email) → appropriate error responses.
        - Invalid login (wrong password, unregistered email) → authentication failure.
    - Protected Endpoints:
        - Access API endpoints (e.g., POST /alerts) without a token → denied.

        - Access with a token but insufficient role (e.g. Victim trying to post Alert) → denied by Firestore rules.
4. End-to-End Scenarios
    - Help Request Workflow:
        - Victim submits request → Volunteer sees it within seconds on dashboard.
    - Resource & Alert Updates:
        - Volunteer posts Resource; another user's view updates automatically.
        - Admin posts Alert; all clients receive it via real-time listener.
5. Performance & Reliability
    - Populate Firestore with dozens of dummy requests, resources, and alerts; verify UI responsiveness.
    - Test operation under simulated intermittent connectivity, confirming offline queuing and sync.

Through these tests we validated that all key backend features-especially the newly implemented Firebase Auth flows-work reliably, securely, and as designed.

History of Work, Current Status, and Future Work :

Securely handle user data and registration: This was a milestone and feature we wanted to implement since the project proposal. It was a key idea for us. We planned this out. And for deliverable 1, we set up a backend with the database to handle user data securely. After that, for deliverable 2, we fully integrated this with the frontend so when users sign up and sign in, their information is securely and safely stored in our database.

Protect general data and financial information of donors: This was planned for in our project proposal. In the project requirement documentation, we outlined what needed to happen for this to work. Afterwards for deliverable 1, we had a form where users could just input an amount of money and click donate. This was a placeholder for what we would set a milestone for in deliberante 2. We implemented a Stripe checkout where when users would want to donate money, they would get redirected to a secure and well known payment processor. This was done through Stripe. Using Stripe, we successfully can assure our users that when they use their credit card to donate money, their information is processed safely.

Make sure victims get the help they need: The main point of our website from the proposal was to streamline help with the use of AI Matching. We planned how we were gonna do this in the proposal and elaborated on it for the project requirement documentation. In deliverable 1, our backend programmer Kevin focused heavily on this feature. Afterwards in deliverable 2, we implemented this into the frontend when a user requests help, the system automatically streamlines help.

Be able to provide real time updates about the unfolding situation: Another key feature since the proposal is being able to give the users of the website and victims a real time update on the ongoing situation. This is key as it would help those in need stay alert and others to be on high guard as well. In deliverable 1, we had this information uploaded to our database. It would then populate the screen whenever there was a disaster alert. In deliverable 2, we elaborated on this feature by having the backend continuously refresh and update. This allows users to get the most real time updates as it comes onto the channel.

Being able to donate: Donations is a key feature as a website would not survive without the help of the general public. This was a must since the proposal. In deliverable 1, we

created the screen for this and a form. We also had the database populated to show past donations. After the deadline we knew we had to make it better by having a way for people to donate. This is where we implemented Stripe Checkout. This also allowed us to securely process users' information. With this feature, our donation screen was complete and more robust than we thought when we planned this out.

View resources: Another key milestone since proposal, this would allow anyone to see what resources were available and where they could be obtained. We populated our database to display the resources and their locations for deliverable 1. But that wasn't enough for us. We improved how the data came in afterwards. We also added certain filters like sorting by quantity or by city. We also added a search bar where users could search for the needed resources they are looking for directly. This feature was something we added after deliverable 1. This made the system completely more robust and allows the users to find the resources they need more effectively.

Key Accomplishments :

- Created a unique home screen where all key functions are displayed for easy access
- Implemented user stories and contact information at the bottom of the home screen
- Implemented a resource inventory screen where users can see available inventory by quantity and city. Allows filtering and searching for quick location of resources. Can also add a resource.
- Implemented an emergency alerts screen where emergency alerts are posted real time for users to be informed.
- Implemented a request help screen where users can request for certain aid. This then gets AI matched for the nearest and most effective volunteer. You are able to see current requests on the screen and on the given map for easy location for better help.
- Implemented a donations screen where users can securely and safely donate money and resources. If using a credit card, users will be sent to Stripe Checkout for secure handling of users data. Users can also pass donations made to the cause.
- Implemented a secure Stripe Checkout processing system
- Added a safe way for users to sign in / sign up
- AI Matching to get victims the fastest help
- Learned how to code using Flutter for frontend

- Learned about APIs and Firebase for storage
- Version Management
- Teamwork
- Project Management

Future Work :

- Upload the website using firebase
- Improve AI Matching
- Add more key features such as profiles, organizational help and information.
- UI Improvement
- Database handling
- Continuous Testing
- A complete and secure website
- How to track volunteers when matched with a victim (see how far they are from your location)

References :

UML Diagrams - UML.drawio : for designing UML Diagrams. The tool is useful for designing and documenting different aspects of the disaster relief platform.

ScienceDirect - Crowdsourcing in Disaster Relief: Database and research. A large article about the uses of Crowdsources and how it is helpful. Discusses how crowdsourcing technologies can be leveraged to enhance disaster response. It provides insights into best practices, challenges, and case studies that inform the development of the proposed disaster relief platform.

Global Disaster Preparedness Center - Case Studies: The Global Disaster Preparedness Center (GDPC) is an initiative focused on improving disaster response and preparedness worldwide. Case studies from GDPC provide real-world examples of how different regions and organizations have managed disaster response, offering valuable insights into best practices for developing an effective relief platform.

Google Maps API Documentation: The Google Maps API allows developers to integrate interactive maps, geolocation services, and real-time tracking into applications. This documentation provides guidelines on how to use mapping services effectively, which is crucial for pinpointing affected areas, locating resources, and guiding volunteers and first responders.

Twilio SMS API Documentation: Twilio provides cloud-based communication services, including SMS, voice, and video messaging. The Twilio SMS API allows the disaster relief platform to send real-time alerts, emergency messages, and updates to victims, volunteers, and organizations. This documentation explains how to implement SMS-based notifications.

FastAPI Documentation: FastAPI is a modern web framework for building APIs with Python. It is known for its speed, efficiency, and ease of use. The FastAPI documentation provides details on creating and managing APIs, which are essential for connecting different components of the disaster relief system, such as mobile apps, databases, and external services.

Stripe Checkout: Allows users to use stripes to safely and securely make donations to our cause. It is safely encrypted and one of the most known online payment systems out there. It allows you multiple ways to make your payment.

Reflective Essays :

Casey :

For this project, I took on a leader / project manager role. This was my first time trying to take charge of a project and it came with many different experiences that will be good to learn from. I wanted to be able to get the work and project done in a timely manner while staying responsible. Those are just a few traits of being a project manager. I also worked and developed heavily in the frontend, working on the user interface alongside my teammate Andy. Together we put out a user interface that I am quite proud of and will be able to look at and share with my peers. Originally I was planning to do more backend work but a few challenges came up that altered our group's original plan. I will get to this soon.

Technical challenges I faced were plenty. During project proposal time, it was proposed that we use flutter to build the website. I had no idea how to use flutter to begin with. I had no prior experience. So building the frontend with a language I was not familiar with was challenging and took lots of time to install and learn how to use. Hours of youtube tutorials were watched just to be able to put out a screen and a widget. Another technical challenge was figuring out how to connect front and backends using flutter. The database was firebase and that was also new to me. But that was easier to learn for sure. A prerequisite I feel like should be covered would be how to actually build a website or app. Thankfully I have built websites and apps before this course so that helped me to start. But in prior UTD classes, I did not have a single project where I was taught or tasked to build a simple website or app. A simple thing to be taught is how to make the most basic screen and how to connect both ends of the project.

The things I learned in the course that were most helpful were the project management skills / chapter we covered. Like I said this role was new to me and having this taught in class helped a lot. I learned what it takes to be a project manager and to be a good one. I wanted to set deadlines and to follow them. I wanted to stay on track and that happened for the most part. Another technique was to oversee issues regarding the project. Whether it was with the code or the group, I wanted to be able to solve it in a way a project manager would. Another thing that helped me was the ethics of software engineering. There are plenty of codes we have to follow to gain respect and trust among peers. I felt like knowing these ethics helped our trio get along and to complete the project in time.

The technique that was most useful in my opinion is the project management aspect of software engineering. Not only for me but for my teammates as well. They understood the role and what came with it. So everyone knew what it took to finish the website and have it actually be good and worth something. The least useful technique I would say would be project planning just in this aspect as we worked seperate for our part for the most part. It would help if we would see eachother everyday but this is not the workspace for it. This was a semester-long project so that would be impossible. Working informally made that technique a little less useful.

Some challenges were making sure the frontend was developed cleanly. I did not know how to use flutter and make screens and what comes with flutter. So learning to flutter was challenging. Another one was a certain teammate that is not listed in this final report. It was not Kevin or Andy. But he did not do his part in this semester-long project. We gave him many chances and he just put us and this project off. This caused our group to pick up the slack and change the dynamics of our project. I moved primarily to the frontend while Kevin had to pick up the slack in the backend. It gave us more work but we ended up being able to manage it.

There were plenty of benefits. I learned how to be a leader as part of this group. I experienced learning how to manage time and the amount of work required to meet deadlines I set. I also grasped the common and intermediate levels of flutter web building. I know what it takes to create a dynamic and working user interface for an app. This is crucial if a company uses flutter or if I want to do another personal project in the future. Another one is the usage of github. Using git and github I got more hands-on experience with it. Especially when working in a team. Examples are looking at code from others code or pulling code to check others work.

Other knowledge that helped me was my existing coding skills and website work on other platforms. This gave me the basics on how to start and how to organize files. I knew I needed to have multiple screens, widgets, etc. So setting up the foundation of the user interface gave me a leg up and saved me some time building the frontend. Another tip was being someone who teammates can trust. I put myself out there as responsible and hard working. This in return rubbed off on others to do their part and to recognize my work as well. This boosted team morale and in the end helped us finish the project in a timely manner.

Overall this course taught me a lot. It taught me what is expected of the industry when you go work on a project. The project itself was mostly self learning. I feel like I learned to build the entire website by myself. This is where the course fell short for me as I would like assistance with website building as well if it is expected of us. Other than that, the techniques taught by Professor Feng were great. I enjoyed listening to her lectures about essential software engineering ethics and techniques. This in return helped me learn how to manage my project and for my future projects ahead. In the end this taught me how to use flutter and how to manage a simple project in a group. That alone is priceless in my opinion. Thank you to the Professor and TA's for the feedback this entire semester and the skills I gained from all of you.

Kevin :

As a backend and AI developer, this project challenged me to design and build essential services with real-world constraints. The most difficult technical challenge was the AI matching system: I decided on a distance-based KNN solution, integrated geolocation using the Geopy library, and optimized queries to Firestore. Learning how to implement Firebase Authentication and Stripe for secure login and payment enhanced my understanding of cloud services and API security. These technologies were not covered in-depth by the course, so I learned them on my own through official documentation and hands-on experimentation.

Teamwork was also a learning experience. Early on, unsynchronized efforts created integration discrepancies, front-end calls did not always align with backend endpoints. We corrected this by adopting brief daily check-ins and requiring GitHub pull-request reviews before code was merged. This organized our workflow and reduced last-minute bugs. In retrospect, the introduction of a light agile board (e.g., Trello) from the start would have also simplified task tracking.

Requirements analysis and UML modeling course content was invaluable. By pinning down use cases and domain models upfront, I avoided downstream rework in design. The class diagram exercise directly influenced my Firestore schema: mapping Victims, Volunteers, and NGOs to a single User class with role-based permissions simplified both data structure and security rules. Conversely, formal effort estimation felt less relevant given our tight timeline; we found it more effective to re-estimate tasks weekly than adhere to Gantt charts.

Unit tests for the matching logic, integration tests for database workflows, and Stripe webhook simulation reaffirmed my appreciation for automated testing. That being said, due to the time component, we employed manual end-to-end scenarios for several features. In future iterations, I would prioritize test automation earlier on.

I learned about leadership and flexibility from this team project. Leadership of the backend work meant owning failures - at one point, a misconfigured Firestore rule broke request creation, which I fixed by reconfiguring the rule and adding a test. Those experiences reinforced the importance of logging, descriptive error messages, and robust security configurations.

Overall, the course's structured approach, from proposals to deliverables, provided a solid foundation for progress. To improve future classes, I'd suggest brief modules on popular frameworks (e.g., Firebase, RESTful APIs) and basic DevOps practices (CI/CD, deployment). These additions would better prepare students for full-stack implementation.

In summary, this project not only consolidated my technical capabilities-particularly in backend development on the cloud and algorithm design-but also reinforced the importance of clear communication, iterative planning, and collaborative problem-solving. These are lessons that will inform my attitude towards tackling software engineering challenges in the future.

Andy :

In this course, I worked primarily on the frontend development of our group project. Through this experience, I learned a lot about the practical application of software engineering techniques such as system modeling, architectural design, configuration management, and agile development. Working in Flutter, I helped build the main user interface screens including the homepage, request posting screen, donation screen, resource inventory, profile screen, and emergency alerts. Overall, I felt the course provided a solid foundation in the major steps of the software engineering process, though I think it could have spent a little more time diving deeper into frontend-specific best practices.

One of the biggest technical challenges I faced was figuring out how to organize and scale the frontend code as more screens and features were added. It was also difficult initially to properly manage asynchronous data fetching and update the UI without bugs. In terms of topics that could have been covered in a prerequisite, I think a basic introduction to real-world frontend architecture and asynchronous programming would have been really helpful before jumping into a project like this.

The software engineering techniques that helped the most were the emphasis on agile development and system modeling. Being able to incrementally build and refine the frontend over time made it much easier to adapt to changes. The focus on use case diagrams and thinking about user flow early also helped guide how I structured the different screens. Configuration management with Git was essential too, even though we ran into occasional merge issues. On the other hand, some techniques like formal risk management and project scheduling felt less useful for us in practice, since our group operated more informally through Discord and text messages rather than full formal planning.

A major team challenge we faced was when one of our members disappeared during the middle of the semester and stopped contributing to the project. This added extra workload and stress to the rest of us, especially as deadlines approached. Individually, it was difficult to keep coordination smooth without regular check-ins, but it also forced me to take more initiative and ownership of my work. On the positive side, working as part of a team made it possible to split responsibilities and get feedback quickly on different parts of the project, which made a big difference.

If I could do the project again over another semester, I would recommend more frequent and structured team meetings, along with clear backup plans in case a teammate drops off. I also believe having stronger backend knowledge earlier would have helped me better integrate the frontend with the backend without relying heavily on the other team members.

Overall, I feel like the course met my needs by giving me a real chance to apply software engineering concepts to a full project. It helped me better understand how planning, design, coding, and teamwork all connect. I do wish there had been slightly more detailed guidance on frontend development practices, but otherwise, the course successfully helped prepare me for future software projects.