Architectural Design Description

This document presents the architecture and detailed design for the software for the SOS project. SOS is a web application providing easy access to federal and local resources for those affected by the LA fires. Right now, information about disaster relief is not very organized. Federal resources are presented in a spreadsheet that is complex and difficult to navigate; local resources, in addition, are scattered across many websites. Things are not centralized. Therefore, people in crisis may have trouble getting help. Additionally, people who want to help may have trouble finding places to donate or volunteer.

System Objective

The objective of this application is to connect people affected by the fires or who want to help with appropriate resources. A centralized site for all available resources organized into tabs, categories and filtering system to meet individuals needs.. Additionally, users can quickly find live listings of people offering resources like rooms, food, supplies, and services.

Hardware, Software, and Human Interfaces Section

1. User Authentication
    - Profile Creation (google auth)
    - Profile will hold information: (Name, bio, needs)
    - Database will keep track of all profile creations
2. Profile Listings
    - All the profiles created will be posted on a listing
    - Clicking interaction, all profiles can be clicked on for more details
3. Resources Listings
    - Several tabs for all the types of resources
    - Each tab will be a listing of all the resources for that category, which will come from the database of resources
    - Each resource will have data: (name, location, website, phone number)
4. Matching Algorithm
    - Algorithm to match profiles with resources that best match needs
5. Recommended Listings
    - All closest matched resources will be on this listing page
6. Filtering Feature
    - In each listing page there will be a filtering icon users can interact with to sort info by federal, local, state, location, and recency.

Architectural Design Section

Major Software Component Section

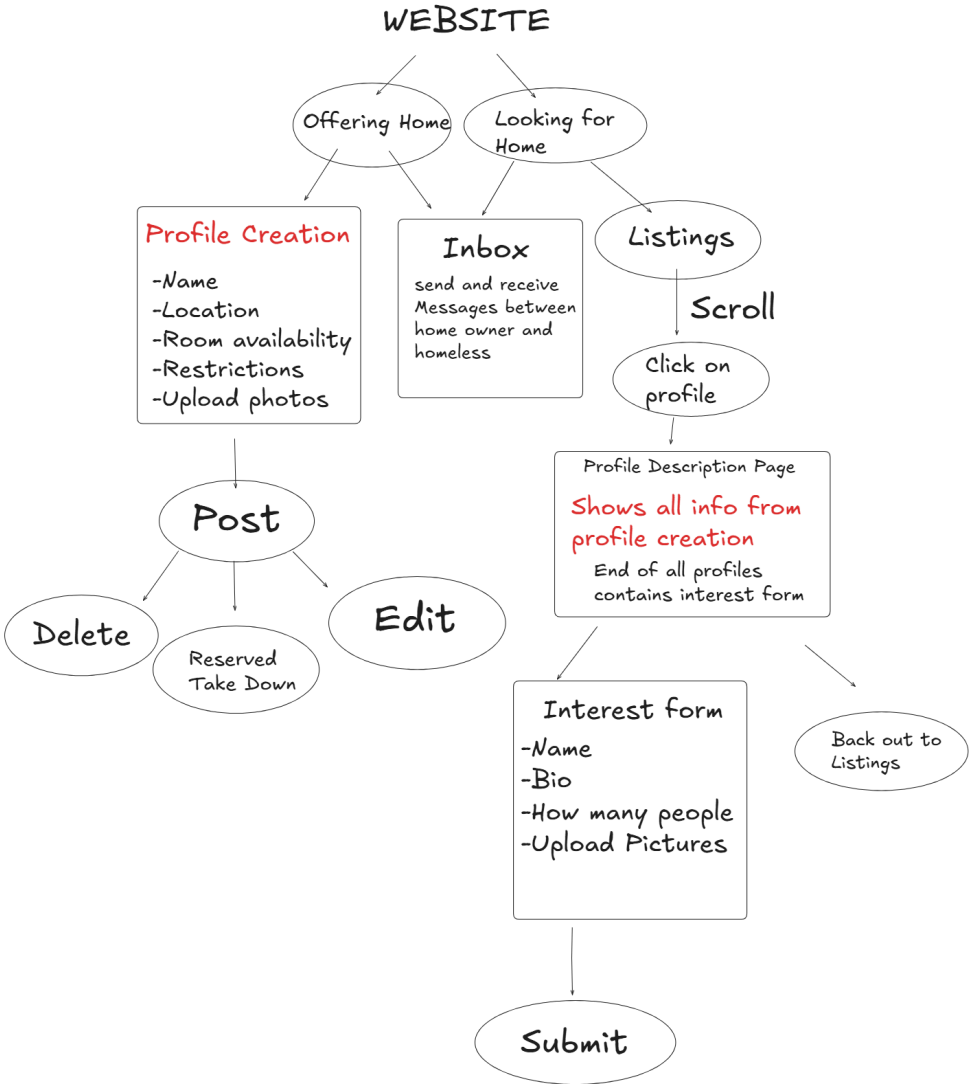
Major Software Interaction Section

Profile Creations will interact with the database to keep track of all of them. And that will interact with each individual page by posting profiles in the correct pages/sections.

Matching system will interact with both the user's profile and other user profiles/resources to recommend matching resources.

Filtering system will also interact with contents on each page and be able to correctly output what users want ot see

Architectural Design Diagrams Section



Detailed Description:

CSC and CSU Description Sections

4. Detailed Design

4.1 Computer Software Configuration Item (CSCI): FireAid Web Application

The **FireAid Web Application** is a full-stack web app designed to assist victims of the LA fires in accessing vital resources, while also providing a platform for donors and volunteers to contribute. The system is developed using Next.js for the frontend and Supabase for backend services including authentication and a relational database.

4.2 Computer Software Components (CSCs)

Each CSC is composed of one or more Computer Software Units (CSUs) that fulfill a specific functional purpose within the application.

4.2.1 CSC: User Management CSC

Handles all operations related to user authentication, registration, profile management, and account security.

CSUs:

- `AuthService`: Handles login, signup, password reset, and session management (interface to Supabase auth).
- `ProfileManager`: Allows users to view and update their profiles.
- `UserContext`: Provides global user state throughout the application using React Context API or Zustand.
- `AccessControl`: Manages role-based access and permissions.

4.2.2 CSC: Resource Directory CSC

Manages the storage, retrieval, and display of resources in the Supabase database.

CSUs:

- `ResourceModel`: Defines the structure and types for resource entries (name, location, description, link, category, etc.).
- `ResourceService`: Provides CRUD operations (create, read, update, delete) for resources via Supabase client.
- `FlagService`: Allows users to report outdated or broken resource links.
- `UploadForm`: UI component for users to submit new resources.
- `ResourceListView`: Displays resources and integrates filtering/search.

4.2.3 CSC: Search and Filter CSC

Facilitates dynamic filtering and keyword searching for resources.

CSUs:

- `SearchBar`: UI input that triggers keyword-based search queries.
- `FilterPanel`: UI component that allows users to select filters (e.g., location, category, proximity).
- `FilterService`: Processes filters and search queries to retrieve relevant data from the backend.
- `LocationService`: Handles geolocation and distance-based proximity filtering.

4.2.4 CSC: Navigation and Interaction CSC

Manages the navigation and UI flow based on user intent (e.g., Give vs. Receive resources).

CSUs:

- `Navbar`: Main navigation bar allowing mode selection and access to different app sections.

- `ModeContext`: Maintains current user intent ("giving" or "receiving") across components.
- `RouteGuard`: Redirects or limits access based on auth state or selected mode.

4.2.5 CSC: Matching Engine CSC

Implements logic to recommend resources tailored to individual user needs.

CSUs:

- `MatchService`: Core logic to analyze user profile and preferences to recommend resources.
- `UserPreferenceModel`: Stores user-stated preferences (e.g., categories of interest, location).
- `RecommendationList`: Displays the recommended resources to the user.
- `FeedbackLoop`: Allows users to rate relevance, feeding back into future matches.

6.3 Class Descriptions

6.3.1 Introduction

The following sections provide the details of all classes used in SOS/LINK. Each class plays a specific role in supporting the system's goals of providing disaster resources and facilitating donation and volunteer efforts. The classes span authentication management, user profile handling, resource upload and retrieval, filtering, navigation, and personalized matching. The smaller utility and model classes are introduced first, followed by progressively larger and more complex service and UI classes.

6.3.1.1 Class: `UserPreferenceModel`

Purpose:
Stores the user's preferences used in the resource matching engine.

Fields:

- `userId` (string): The ID of the user this preference model belongs to.
- `preferredLocations` (string[]): List of cities or regions the user is interested in.
- `preferredCategories` (string[]): Types of resources (e.g., housing, food, wellness).
- `maxDistanceKm` (number): Maximum distance (in km) for nearby resources.

Methods:

- `constructor(data: object)`: Initializes the model with user preference data.
- `updatePreferences(newPrefs: object)`: Updates stored preferences with new data.
- `toJSON()`: Returns a plain object representation for database storage.

6.3.1.2 Class: `ResourceModel`

Purpose:
Represents a resource listing stored in Supabase.

Fields:

- `id` (string): Unique identifier.
- `name` (string): Name/title of the resource.
- `location` (string): Geographic location.
- `category` (string): Resource type (e.g., food, housing).
- `description` (string): Brief overview of what the resource offers.
- `link` (string): URL to the resource.
- `createdBy` (string): ID of the user who submitted it.

- `createdAt` (Date): Timestamp of resource submission.

Methods:

- `constructor(data: object)`: Initializes the resource model.
- `validate()`: Checks that required fields are filled and URL is valid.
- `toJSON()`: Returns the resource as a plain object for API/database interaction.

6.3.1.3 Class: `AuthService`

Purpose:
Manages all user authentication and session handling using Supabase.

Fields:

- `supabaseClient` (SupabaseClient): Initialized client for backend communication.

Methods:

- `login(email: string, password: string)`: Authenticates the user.
- `signup(email: string, password: string)`: Registers a new user.
- `logout()`: Signs the user out and clears session data.
- `resetPassword(email: string)`: Sends password reset instructions.
- `getSession()`: Retrieves the current session from Supabase.

6.3.1.4 Class: `ProfileManager`

Purpose:
Handles retrieval and updating of user profile data from Supabase.

Fields:

- `supabaseClient` (SupabaseClient): Client instance for user profile requests.

Methods:

- `getProfile(userId: string)`: Retrieves a user's profile from the database.
- `updateProfile(userId: string, data: object)`: Updates user fields (name, language, etc.).
- `deleteAccount(userId: string)`: Deletes the user's account and associated resources.

6.3.1.5 Class: `ResourceService`

Purpose:
Handles all interaction with the resource table in Supabase.

Fields:

- `supabaseClient` (SupabaseClient): Used to perform backend operations.

Methods:

- `fetchResources(filters: object)`: Returns all resources matching given filters.
- `addResource(resource: ResourceModel)`: Adds a new resource to the database.
- `deleteResource(id: string)`: Deletes a specific resource.
- `flagResource(id: string, reason: string)`: Flags a resource for admin review.
- `getResourceById(id: string)`: Returns a single resource entry.

6.3.1.6 Class: `MatchService`

Purpose:
Implements logic to match users with relevant resources based on preferences and location.

Fields:

- `resourceService` (ResourceService): Dependency to fetch and score resources.
- `distanceService` (LocationService): Calculates distances between user and resource locations.

Methods:

- `generateMatches(userPrefs: UserPreferenceModel)`: Returns an array of ranked matches.
- `calculateScore(resource: ResourceModel, userPrefs: UserPreferenceModel)`: Uses heuristics to assign relevance scores.
- `getRecommendations(userId: string)`: Fetches user prefs, finds and returns top matches.

6.3.1.7 Class: `SearchBar`

Purpose:
A React component class responsible for keyword-based resource searches.

Fields:

- `query` (string): Text input from the user.
- `onSearch` (function): Callback triggered when the search is submitted.

Methods:

- `handleChange(e)`: Updates the query string in state.

- `handleSubmit(e)`: Prevents form default, calls `onSearch()` with `query`.

6.3.1.8 Class: `FilterPanel`

Purpose:
A React UI component that allows users to filter resources by location, category, and distance.

Fields:

- `selectedLocation` (string): Selected geographic area.
- `selectedCategories` (string[]): Selected resource types.
- `distance` (number): Chosen max distance.
- `onFilterChange` (function): Called when filter values update.

Methods:

- `handleLocationChange(value)`: Updates the selected location.
- `handleCategoryToggle(category)`: Toggles a category.
- `handleDistanceChange(value)`: Updates the distance.

6.4 Interface Descriptions

6.4.1 Introduction

The following sections describe the interfaces used in SOS/LINK. These interfaces define how various Computer Software Units (CSUs) exchange data and invoke one another's functionality. This includes data flow between services and UI components, user input triggers, API calls, and how application state is shared or controlled across subsystems. These interfaces are critical to ensuring modularity, maintainability, and clear separation of concerns in the software.

6.4.1.1 Interface: `User Authentication Interface`

Interacting CSUs:

- `AuthService`
- `ProfileManager`
- `UserContext`
- UI Pages (Login, Signup, ForgotPassword)

Purpose:
Handles user authentication processes, maintains login sessions, and provides user context throughout the application.

Input/Output:

- Receives email and password input from UI.
- Returns session tokens, authentication success/failure messages.
- Emits user context changes that UI can subscribe to.

Data Format:

```
{
  "email": "user@example.com",
  "password": "securePassword123"
}
```

Communication Methods:

- RESTful API calls through Supabase client
- React Context API for session sharing
- Redirects or guarded routes based on authentication state

6.4.1.2 Interface: `Resource Upload and Retrieval Interface`

Interacting CSUs:

- `UploadForm`
- `ResourceService`
- `ResourceModel`
- `Database (Supabase)`

Purpose:
Facilitates adding, editing, retrieving, and deleting resource entries in the database. Allows user-submitted data to be structured and validated before storage.

Input/Output:

- Accepts new resource submissions via form.
- Sends formatted data to Supabase for storage.
- Retrieves and formats stored resources for UI rendering.

Data Format:

```
{
  "name": "Free Meals at Local Shelter",
  "location": "Los Angeles, CA",
  "description": "Hot meals served daily 12–6 PM.",
  "link": "http://shelter.org",
  "category": "Food"
}
```

Communication Methods:

- Supabase client SDK calls (insert, select, delete)
- Form validation through local logic or Yup schema
- UI reactivity triggered by changes in returned data

6.4.1.3 Interface: `Search and Filter Interface`

Interacting CSUs:

- `SearchBar`
- `FilterPanel`
- `FilterService`
- `ResourceListView`
- `ResourceService`

Purpose:
Allows users to input keywords and filter parameters to refine the resource list. Connects user intent with actual filtered queries.

Input/Output:

- Accepts search keywords and selected filter options.
- Outputs refined resource lists based on current filter state.

Data Format:

```
{
  "query": "housing",
  "location": "Burbank",
  "category": ["Housing", "Supplies"],
```

```
  "distanceKm": 25
}
```

Communication Methods:

- Props and event handlers in React
- Filter queries translated into Supabase filter chains
- State lifted to parent or shared via context/hooks

6.4.1.4 Interface: `Matching Engine Interface`

Interacting CSUs:

- `MatchService`
- `UserPreferenceModel`
- `ResourceService`
- `RecommendationList`
- `ProfileManager`

Purpose:
Fetches user preferences and current resource pool to return sorted and scored matches. Powers the personalized user experience.

Input/Output:

- Accepts `userId` or `UserPreferenceModel`.
- Outputs a sorted list of relevant `ResourceModel` entries.

Data Format:

```
{
  "userId": "abc123",
  "preferredCategories": ["Wellness", "Housing"],
  "preferredLocations": ["Downtown LA"],
  "maxDistanceKm": 30
}
```

Communication Methods:

- Internal scoring logic
- Integration with Supabase data via `ResourceService`
- Outputs JSON array of ranked results to be rendered

6.4.1.5 Interface: `Navigation and Mode Interface`

Interacting CSUs:

- `Navbar`
- `ModeContext`
- All page components (Home, Search, Upload, Profile)

Purpose:
Allows the user to select whether they are looking to **give** or **receive** help, and reflects that intent throughout the app.

Input/Output:

- Receives a mode toggle event from UI.

- Emits current mode to consuming components.

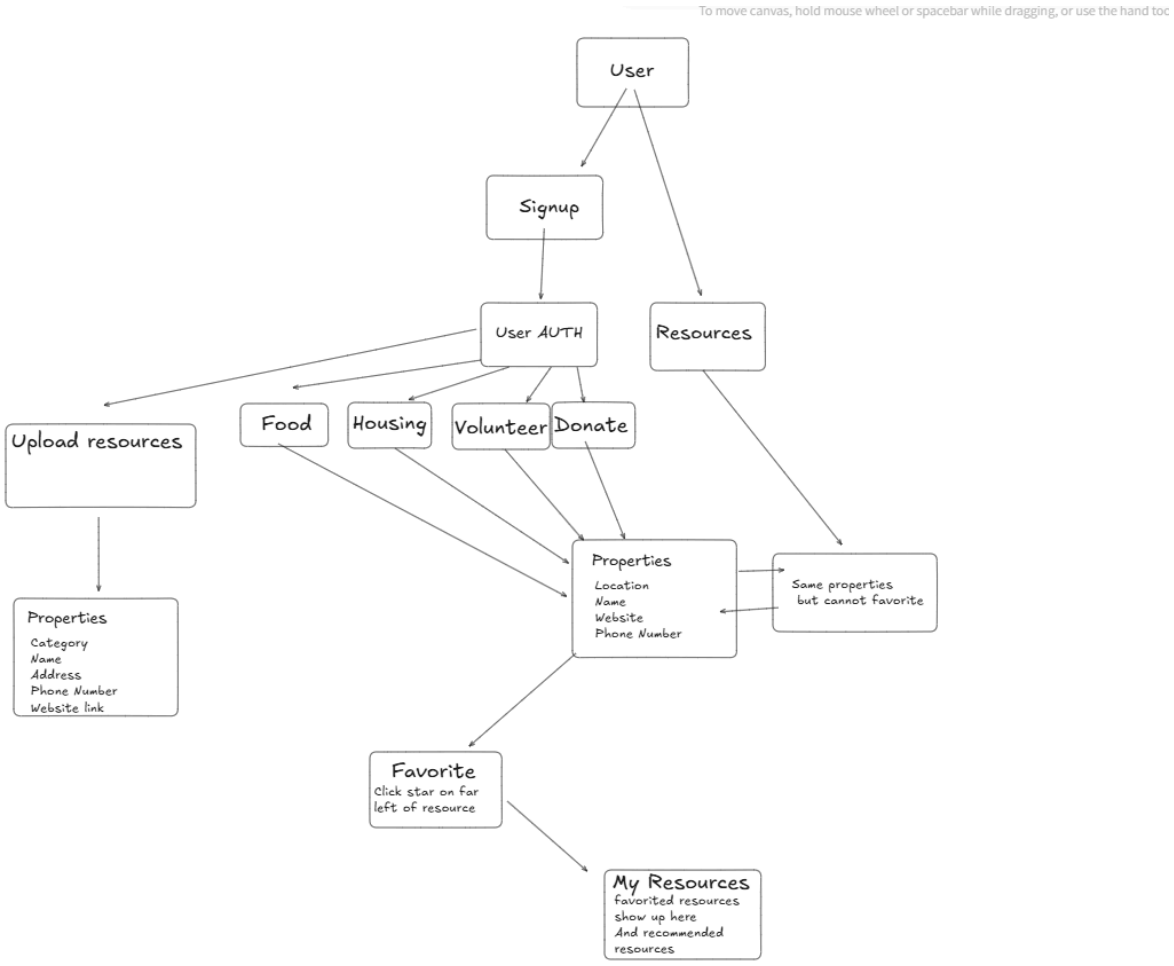
Data Format:

```
{
  "mode": "GIVE" // or "RECEIVE"
}
```

Communication Methods:

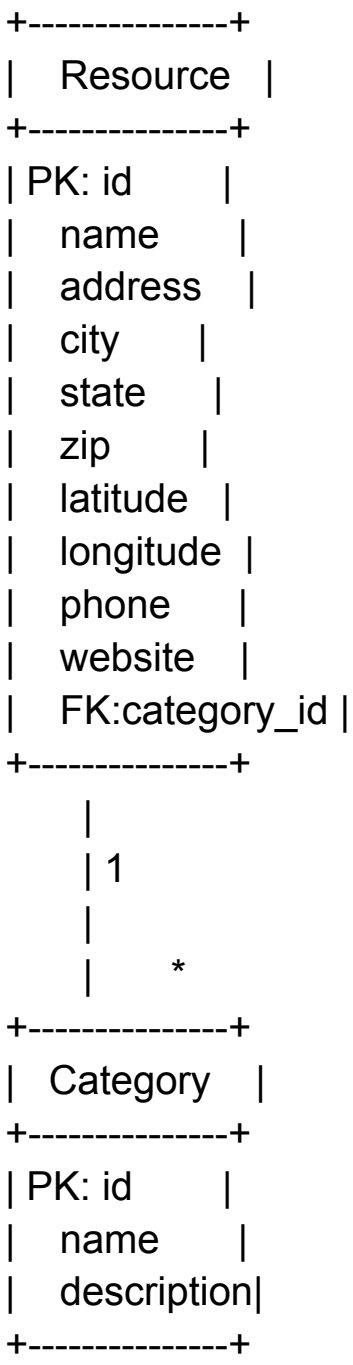
- React Context or Zustand store for global state
- Conditional rendering of components based on mode
- Page access controlled by current intent

DESIGN Diagram



DATABASE DESIGN

Entity-Relationship Diagram (UML-style)



Database Schema Description

The database consists of two main tables:

1. Resource Table: Stores all information about available resources for fire victims
   - Contains geolocation data (latitude/longitude) for distance calculations
   - Has a foreign key relationship to the Category table

2. Category Table: Defines the different types of resources available
   - Pre-populated with your specified categories: food, housing, supplies, wellness, donation, and volunteer

Database Access Pattern

The database will be accessed primarily through these operations:

1. Resource Retrieval by Location: When a user selects a category, the application will:
   - Get the user's current location (via browser geolocation API or entered address)
   - Query the database for resources in the selected category
   - Calculate distances using the Haversine formula with stored latitude/longitude
   - Return results ordered by proximity

2. Administrative Operations:
   - CRUD operations for resource management (admin interface only)
   - Category management (rarely modified after initial setup)

The frontend will make API calls to backend services that handle the database queries, keeping the database connection secure and not exposed to client-side code.

Database Security Implementation

1. Access Control:
   - Read access for all users (resource listings)
   - Write access restricted to authenticated admin users only
   - Role-based access control implemented at application level

2. Data Protection:
   - All database connections use parameterized queries to prevent SQL injection
   - Sensitive admin operations require JWT authentication
   - Database credentials stored in environment variables, not in code

3. Infrastructure Security:
   - Database hosted on secure cloud provider with firewall rules
   - Regular automated backups configured
   - Connection encryption (TLS) enforced for all database communications

4. User Privacy:
   - No personally identifiable user information stored in this database
   - Location data used only for distance calculation, not stored persistently