System Architecture - Kilometros de Vida

Overview

Kilometros de Vida is a full-stack MERN application connecting food donors with volunteer drivers to reduce food waste and alleviate hunger.

Technology Stack

Frontend

- **React 19** with Vite for fast development
- **React Router** for client-side routing
- **Tailwind CSS 4** for styling
- **Framer Motion** for animations
- **Axios** for HTTP requests
- **React Leaflet** for map visualization

Backend

- **Node.js** with Express.js
- **MongoDB Atlas** for database
- **Google OAuth 2.0** for authentication
- **Nominatim API** (OpenStreetMap) for geocoding

System Architecture Diagram

```
graph TB
    subgraph "Client Browser"
        A[React App]
        B[React Router]
        C[User Context]
        D[Components]
    end

    subgraph "Frontend Server"
        E[Vite Dev Server :5173]
    end

    subgraph "Backend Server"
        F[Express.js :5050]
        G[Auth Middleware]
        H[API Routes]
    end

    subgraph "External Services"
        I[MongoDB Atlas]
```

```
        J[Google OAuth]
        K[Nominatim API]
    end

    A --> E
    E --> F
    F --> G
    G --> H
    H --> I
    F --> J
    A --> K

    style A fill:#61DAFB
    style F fill:#68A063
    style I fill:#4DB33D
```

Data Flow

Authentication Flow

```
sequenceDiagram
    participant U as User
    participant C as Client
    participant G as Google OAuth
    participant S as Server
    participant DB as MongoDB

    U->>C: Click "Sign in with Google"
    C->>G: Request authentication
    G->>U: Show Google login
    U->>G: Enter credentials
    G->>C: Return ID token
    C->>S: POST /api/auth/google {token}
    S->>G: Verify token
    G->>S: Token valid + user info
    S->>DB: Upsert user record
    DB->>S: User document
    S->>C: Return user data
    C->>C: Store in UserContext
```

Donation Creation Flow

```
sequenceDiagram
    participant U as User
    participant C as Client
    participant S as Server
    participant DB as MongoDB
```

```
    participant N as Nominatim API

    U->>C: Fill donation form
    U->>C: Submit form
    C->>S: POST /join/giver {donation data}
    S->>S: Validate input
    S->>DB: Insert donation
    DB->>S: Donation created
    S->>C: Success response
    C->>N: Geocode address
    N->>C: Lat/Lng coordinates
    C->>C: Cache in localStorage
    C->>C: Update map
```

Component Hierarchy

```
graph TD
    A[App.jsx] --> B[Navbar]
    A --> C[Routes]
    A --> D[Footer]

    C --> E[Home]
    C --> F[About]
    C --> G[Volunteer]
    C --> H[Giver]
    C --> I[Contact]
    C --> J[Profile Protected]

    B --> K[Login]
    E --> L[DonationMap]

    J --> M[ProtectedRoute]

    style M fill:#FFD700
    style K fill:#90EE90
    style L fill:#87CEEB
```

API Endpoints

Authentication

Method	Endpoint	Description	Auth Required
POST	`/api/auth/google`	Verify Google token and create/update user	No

Donations (Givers)

Method	Endpoint	Description	Auth Required
POST	`/join/giver`	Create new donation	No*
GET	`/api/data`	Get all donations (public)	No
GET	`/api/my-donations`	Get user's donations	Yes
PUT	`/api/donations/:id`	Update donation	Yes
DELETE	`/api/donations/:id`	Delete donation	Yes

*Links to user if logged in

Volunteers (Drivers)

Method	Endpoint	Description	Auth Required
POST	`/join/driver`	Register as volunteer	No*
GET	`/api/my-volunteer-shifts`	Get user's shifts	Yes

Database Schema

Collections

users

```
{
  _id: ObjectId,
  email: String (unique),
  name: String,
  picture: String (URL),
  lastLogin: Date
}
```

givers (donations)

```
{
  _id: ObjectId,
  userId: String (optional, links to user),
  orgName: String,
  contactPerson: String,
  donorEmail: String,
  donorPhone: String,
  foodType: String,
  pickupTime: String,
  address: String,
  createdAt: Date
}
```

drivers (volunteers)

```
{
  _id: ObjectId,
  userId: String (optional),
  volunteerName: String,
  volunteerEmail: String,
  volunteerPhone: String,
  availability: String,
  createdAt: Date
}
```

Security Measures

Implemented

1. **Google OAuth 2.0**: Secure authentication without password storage
2. **Environment Variables**: Sensitive data in `.env` files
3. **Protected Routes**: `ProtectedRoute` component prevents unauthorized access
4. **Input Validation**: Server-side validation for all form submissions
5. **CORS**: Configured to allow frontend-backend communication