# GatherTown Project Documentation

## Overview

GatherTown is a community-driven local event finder platform designed to simplify event discovery and management through advanced search, authentication, and role-based management features. It provides comprehensive event management capabilities for organizers and seamless browsing for participants. The motivation behind the project is to create a user-friendly platform where people can easily find and participate in local events, leveraging geolocation and an interactive map interface.

## Table of Contents

1. [Project Inception](#project-inception)
2. [Backend Development](#backend-development)
   * Project Setup
   * Authentication and Authorization
   * Event Management
   * Geolocation and Mapping
3. [Frontend Development](#frontend-development)
   * User Interface
   * Event List and Map Integration
   * Search and Filter Features
4. [Database](#database)
   * Schema Design
   * Event Data Generation
5. [Deployment and Testing](#deployment-and-testing)
6. [Known Issues and Fixes](#known-issues-and-fixes)
7. [Future Enhancements](#future-enhancements)
8. [Installation](#installation)
9. [Configuration](#configuration)
10. [Usage](#usage)
11. [Directory Structure](#directory-structure)
12. [API Documentation](#api-documentation)
13. [Contributing](#contributing)
14. [License](#license)

## Project Inception

The GatherTown project was conceived to provide a community-driven local event finder, aiming to simplify event discovery and management by leveraging geolocation and an interactive map interface. The project's inception involved brainstorming key features, designing the architecture, and selecting the technology stack.

## Backend Development

### Project Setup

1. **Initial Setup**: The backend environment was set up with settings including Database URI, JWT Secret, and Server Port. Folders were configured for Config, Utilities, Integration Tests, and CI/CD pipelines.
2. **Technology Stack**: The backend was built using Node.js, Express, and MongoDB. Authentication was handled using JWT, bcrypt for password hashing, and OAuth tokens.

### Authentication and Authorization

1. **User Registration and Login**: Implemented endpoints for user registration and login, using bcrypt for password hashing and JWT for session management. Additional features included email verification and password reset.
2. **Role-Based Access Control**: Ensured secure access to different parts of the application based on user roles, implemented using middleware that checks user roles before granting access to specific routes.

### Event Management

1. **Event CRUD Operations**: Developed endpoints for creating, updating, deleting, and fetching events. Events include fields such as title, description, date, location, category, and images.
2. **Event Filtering**: Implemented the /api/events/nearby endpoint to filter events based on user location.

### Geolocation and Mapping

1. **Geolocation API Integration**: Integrated geolocation to fetch user coordinates and filter nearby events. Switched from Google Maps API to Mapbox due to cost concerns.
2. **Reverse Geocoding**: Implemented reverse geolocation for event addresses, allowing users to see event locations in a human-readable format.

## Frontend Development

### User Interface

1. **React Setup**: The frontend was developed using React. Components were created for user authentication, event listings, event details, and map integration. Design principles included a responsive layout and intuitive navigation.
2. **Interactive Homepage**: Designed the homepage with sections for event lists, detailed views, and an interactive map, ensuring a seamless user experience.

### Event List and Map Integration

1. **Event List**: Displayed events with details, a "load more" button, and a footer showing the number of events displayed. Handled pagination and dynamic updates based on user interactions.
2. **Map Integration**: Integrated Mapbox to display event locations on a map. Addressed issues such as slow loading and multiple script inclusions, ensuring a smooth and interactive map experience.

### Search and Filter Features

1. **Search by Location**: Allowed users to search events by entering a zip code or city. Implemented dynamic event filtering based on the map view and user location.
2. **Search by Title and Category**: Enhanced search functionality to filter events by title, category, city, state, zip code, and date ranges.

## Database

### Schema Design

1. **Event Schema**: Designed a comprehensive schema for events, including fields for title, description, date, location (coordinates, city, state, zip code, country), category, time, creator, and images.
2. **User Schema**: Created a schema for user data, including fields for username, password, email, and roles.

### Event Data Generation

1. **Data Generation Script**: Developed a Python script using Faker, geopy for geocoding, and Cloudinary for image uploads to generate realistic event data.
2. **Data Import**: Imported generated data into MongoDB using MongoDB Compass, ensuring the database is populated with relevant and accurate event information.

## Deployment and Testing

1. **Testing**: Used Jest for unit and integration tests. Ensured a clean database state for each test run. Fixed various issues related to event fetching, map integration, and error handling.
2. **Deployment**: Configured CI/CD pipelines for continuous integration and deployment. Ensured secure and reliable deployment to the production environment, using platforms like Heroku or AWS.

## Known Issues and Fixes

1. **Google Maps API Key Issues**: Encountered and resolved issues related to invalid Google Maps API keys by switching to Mapbox.
2. **CSS Conflicts**: Resolved conflicting CSS styles in App.css and HomePage.css files, causing layout issues.
3. **Duplicate Keys**: Fixed issues related to duplicate keys in the event list rendering.
4. **Event Fetching**: Addressed issues with the frontend unable to fetch events, resulting in a 404 error.

## Future Enhancements

1. **Real-Time Updates**: Implement real-time updates for event listings using WebSockets.
2. **Advanced Filtering**: Enhance filtering options to include more categories and custom date ranges.
3. **User Profiles**: Develop user profiles to track attended events and preferences.
4. **Mobile Application**: Create a mobile version of the application for iOS and Android.

## Installation

1. Clone the repository:

git clone https://github.com/equansa00/GatherTown.git

cd GatherTown

1. Install the dependencies:

npm install

## Configuration

1. Set up the environment configuration by creating a .env file in the project root:

touch .env

1. Add the required environment variables:

MONGODB\_URI=mongodb://localhost/yourdbname

JWT\_SECRET=yourjwtsecret

## Usage

1. Start the server:

npm start

or, for development:

npm run dev

The API server will run on [http://localhost:5000](http://localhost:5000/).

## Directory Structure

The main folders and files are organized as follows:

.

├── config/ # Configuration files

├── controllers/ # Logic for handling API requests

├── helpers/ # Helper functions and utilities

├── middleware/ # Middleware for request handling

├── models/ # MongoDB models (schemas)

├── routes/ # API routes

├── utils/ # Utility functions (e.g., authentication)

├── \_\_test\_\_/ # Tests for the API

├── server.js # Entry point of the application

├── package.json # Project metadata and dependencies

└── README.md # Project documentation

## API Documentation

Detailed documentation of the API endpoints, including request/response formats and parameters, can be found in the project's /docs folder or an associated API documentation site, generated using tools like Swagger.

## Contributing

Contributions are welcome! If you wish to contribute, please follow these steps:

1. Fork the repository.
2. Create a new feature branch:

git checkout -b feature/your-feature

1. Commit your changes:

git commit -m 'Add your feature'

1. Push to your branch:

git push origin feature/your-feature

1. Create a pull request.

## License

GatherTown is released under the MIT License. See LICENSE for more information.