Find Nearby Cafes - Project Documentation

September 2025

Author and Project Details

- **Name**: Anil Mundhe
- **College Name**: JSPM's Rajarshi Shahu College of Engineering, Pune
- **Experience Name**: FrontEnd Developer Intern - AI Unika Technology, Pvt. Ltd, Pune
- **GitHub Project Link**: `http://github.com/Mundheanil84/Find-Nearby-Cafes`
- **Demo Link**: `https://near-cafe.vercel.app/`

1 Project Description

Find Nearby Cafes is an interactive web application that helps users discover coffee shops and cafes across major Indian cities. The application features an intuitive map interface combined with a comprehensive search system, allowing users to either use their current location or explore cafes in specific cities.

1.1 Key Features:

- **Interactive Map**: Built with `Leaflet.js` showing user location and cafe markers
- **Indian City Search**: Search across 14 major Indian cities with smart suggestions
- **Current Location Detection**: Automatic geolocation with permission handling
- **Cafe Information**: Detailed cafe data including ratings, specialties, and hours
- **Responsive Design**: Mobile-friendly interface using `Tailwind CSS`
- **Real-time Filtering**: Dynamic cafe filtering based on selected city/location

1.2 Tech Stack:

- **Frontend**: `React 18 + Vite`
- **Mapping**: `Leaflet.js + React-Leaflet`
- **Styling**: `Tailwind CSS`
- **Data**: Static `JSON` with Indian cafe information
- **Geolocation**: Browser `Geolocation API`

2 Setup and Installation Instructions

2.1 Prerequisites

- **Node.js** (version 14 or higher)
- `npm` or `yarn` package manager
- Modern web browser with geolocation support

2.2 Step-by-Step Setup

2.2.1 1. Clone or Create Project

```
# If starting from scratch
npm create vite@latest cafe-finder -- --template react
cd cafe-finder

# Install dependencies
npm install leaflet react-leaflet
npm install -D tailwindcss postcss autoprefixer
npx tailwindcss init -p
```

2.2.2 2. Install Dependencies

```
npm install
```

2.2.3 3. Replace Default Files

Replace the default files created by Vite with the project files provided in the implementation.

2.2.4 4. Project Structure Setup

Ensure your project structure matches:

```
cafe-finder/
 public/
    cafes.json
 src/
    components/
       Map.jsx
       CafeList.jsx
       LoadingSpinner.jsx
       SearchBar.jsx
    hooks/
       useGeolocation.js
       useCitySearch.js
    services/
       geocodingService.js
    utils/
       cafeData.js
       constants.js
    App.jsx
    main.jsx
    index.css
```

2.2.5 5. Run Development Server

```
npm run dev
```

2.2.6 6. Access Application

Open your browser and navigate to `http://localhost:5173`

2.2.7 Build for Production

```
npm run build
npm run preview
```

3 Test Cases and Running Tests

3.1 Available Tests

The project includes unit tests for data parsing functions located in `src/utils/cafeData.test.js`

3.2 Running Tests

3.2.1 1. Install Testing Dependencies

```
npm install -D @testing-library/react @testing-library/jest-dom jest @testing-library
    /user-event
```

3.2.2 2. Configure Jest (`jest.config.js`)

```
export default {
  testEnvironment: 'jsdom',
  setupFilesAfterEnv: ['<rootDir>/src/setupTests.js'],
  transform: {
    '^.+\\.(js|jsx)$': 'babel-jest',
  },
};
```

3.2.3 3. Create Setup File (src/setupTests.js)

```
import '@testing-library/jest-dom';
```

3.2.4 4. Add Test Script to package.json

```
{
  "scripts": {
    "test": "jest",
    "test:watch": "jest --watch"
  }
}
```

3.2.5 5. Run Tests

```
# Run tests once
npm test

# Run tests in watch mode
npm run test:watch
```

3.3 Test Cases Implemented

File: src/utils/cafeData.test.js

```
import { parseCafeData, testParseCafeData } from './cafeData';

describe('parseCafeData', () => {
  test('should parse valid cafe data correctly', () => {
    const testData = {
      cafes: [
        {
          id: 1,
          name: "Test Cafe",
          latitude: "40.7589",
          longitude: "-73.9851",
          address: "123 Test St",
          rating: 4.5,
          hours: "9-5"
        }
      ]
    };

    const result = parseCafeData(testData);

    expect(result).toHaveLength(1);
    expect(result[0].name).toBe("Test Cafe");
    expect(result[0].latitude).toBe(40.7589);
    expect(typeof result[0].latitude).toBe('number');
  });

  test('should throw error for invalid data structure', () => {
    expect(() => parseCafeData(null)).toThrow('Invalid cafe data structure');
    expect(() => parseCafeData({})).toThrow('Invalid cafe data structure');
  });

  test('testParseCafeData function should work correctly', () => {
    expect(testParseCafeData()).toBe(true);
  });
});
```

3.4 Test Coverage Areas

- ✓ Data parsing and validation
- ✓ Error handling for malformed data
- ✓ Type conversion (string to number for coordinates)
- ✓ Data structure integrity

4 Design Choices and Assumptions

4.1 Architecture Decisions

4.1.1 1. Component Structure

```
// Rationale: Separation of concerns with focused components
App (Container)
          SearchBar (User Input)
          Map (Visualization)
          CafeList (Data Display)
          LoadingSpinner (UI State)
```

4.1.2 2. State Management

- Used React hooks (`useState`, `useEffect`) instead of external state management
- Local state sufficient for current feature set
- Props drilling minimized through component composition

4.1.3 3. Data Flow

User Interaction → State Update → Component Re-render → Map/CafeList Update

4.2 Key Design Choices

4.2.1 1. Indian Cities Focus

- **Choice**: Predefined list of 14 major Indian cities
- **Reason**: Better user experience for Indian audience
- **Alternative Considered**: External geocoding API (rejected for simplicity)

4.2.2 2. Static JSON Data

- **Choice**: Local `JSON` file instead of external API
- **Reason**:
  - Faster development
  - No API dependencies
  - Consistent demo experience
- **Trade-off**: Limited to predefined cafe data

4.2.3 3. Leaflet over Google Maps

- **Choice**: `Leaflet.js` with `OpenStreetMap`
- **Reason**:
  - Free tier availability
  - Better customization
  - No API keys required
- **Trade-off**: Less sophisticated than Google Maps

4.2.4 4. Mobile-First Design

- **Choice**: `Tailwind CSS` with responsive utilities
- **Reason**: Growing mobile usage for location-based apps
- **Implementation**: Flexbox layouts with breakpoints

4.3 Assumptions Made

4.3.1 1. User Permissions

- Assumed users will grant location access
- Fallback UI provided for permission denials
- Default to Mumbai if location unavailable

4.3.2 2. Data Structure

- Cafe data follows consistent schema
- Coordinates are valid and within India
- All required fields present in `JSON`

4.3.3 3. Browser Support

- Modern browsers with ES6+ support
- `Geolocation API` availability
- CSS Grid/Flexbox support

4.3.4 4. Network Conditions

- Reasonable internet speed for map tiles
- `JSON` file size manageable for mobile networks

4.4 Performance Considerations

4.4.1 1. Map Optimization

- Marker clustering considered but not implemented (small dataset)
- Tile loading with error handling
- Zoom level optimization for city views

4.4.2 2. Data Loading

- `JSON` file cached by browser
- Efficient re-renders with `React memoization`
- Debounced search inputs

4.4.3 3. Accessibility

- Semantic HTML structure
- Keyboard navigation support
- Screen reader compatibility

4.5 Scalability Considerations

4.5.1 Current Architecture Supports:

- Adding more cities easily (update `constants.js`)
- Expanding cafe data without code changes
- Integration with real APIs when needed
- Additional filter types (price, ratings, etc.)

4.5.2 Potential Enhancements:

- Backend API for dynamic data
- User accounts and favorites
- Real-time cafe status updates
- Advanced search filters

This documentation provides a comprehensive overview of the project setup, testing approach, and the thoughtful design decisions made during development. The application balances functionality with simplicity while maintaining scalability for future enhancements.