

**The University of Azad Jammu and Kashmir**

**Department of Software Engineering**

OEL

|  |  |
| --- | --- |
| **Course Title:** | Web Design & Development |
| **Course Code:** | SE-3208 |
| **Semester:** | 6th |
| **Session:** | 2022-2026 |

**Bachelor Of Science In Software Engineering (2022-2026)**

**Department of Software Engineering**

# **City Explorer Web Application**

## ****1. Objective****

The objective of this lab project is to design and implement a simple yet powerful web application that integrates an external API to fetch and display real-world data dynamically.  
The **City Explorer** app allows users to search for any city and view details such as its coordinates, country, timezone, and population using the **OpenCage Geocoding API**.

This project demonstrates how frontend and backend components interact to fetch live data and display it responsively on a web page.

## ****2. Tools and Technologies****

* **HTML5** – for page structure
* **CSS3** – for layout and visual design
* **Bootstrap 5** – for responsive UI
* **JavaScript (Fetch API)** – to handle API requests and display data dynamically
* **PHP** – for backend proxy (to keep the API key secure)
* **OpenCage Geocoding API** – to retrieve city and geographical data
* **XAMPP** – local development environment with Apache and PHP

## ****3. System Design and Workflow****

The City Explorer follows a **client-server architecture**:

1. The **frontend (HTML, CSS, JS)** collects the city name entered by the user.
2. A **request** is sent to the backend (api.php), which contains a PHP proxy script.
3. The **PHP script** securely calls the **OpenCage API** using the user’s city input.
4. The **API returns JSON data** containing latitude, longitude, timezone, and more.
5. The **frontend parses** this JSON data and displays it dynamically on the webpage, along with an **OpenStreetMap** embedded map.

**Workflow Diagram:**

User → Frontend (HTML + JS) → PHP Proxy (api.php) → OpenCage API

Response (JSON) ← PHP Proxy ← API ← Frontend (Displayed in Browser)

## ****4. Implementation Details****

### ****File Structure****

city-explorer/

│

├── index.html → Main HTML interface (Bootstrap)

├── style.css → Custom styling

├── script.js → JavaScript logic for API requests

├── api.php → PHP backend proxy for API key security

└── README.md → Documentation and setup instructions

### ****Key Implementation Steps****

* Created a Bootstrap-based interface for the search form.
* Added a fetch() call in JavaScript to send requests to api.php.
* Used api.php to securely forward city queries to the OpenCage API.
* Parsed the JSON response and displayed city details dynamically.
* Embedded OpenStreetMap iframe using the returned latitude and longitude.

## ****5. API Description****

**API Used:** OpenCage Geocoding API

**Endpoint Example:**

https://api.opencagedata.com/geocode/v1/json?q=London&key=YOUR\_API\_KEY

**Parameters:**

* q — City name entered by the user
* key — Your OpenCage API key
* limit — Number of results (default: 5)

**Data Returned:**

* City and country name
* Latitude and longitude
* Timezone
* Population (if available)

## ****6. Results and Output****

### ****Expected Output****

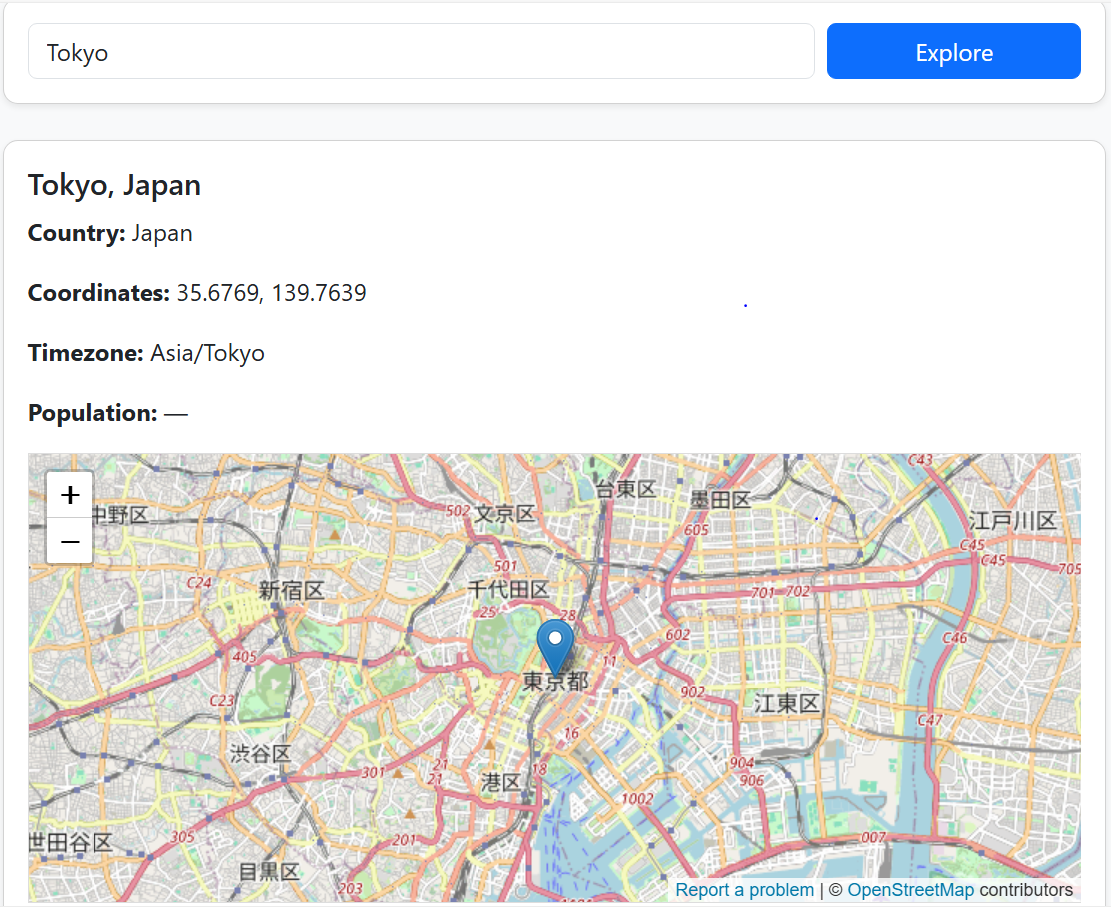
When a user searches for a city:

* The page displays:
  + City name and country
  + Coordinates (latitude & longitude)
  + Timezone
  + Population (if available)
  + A map centered on the city’s coordinates

### ****Sample Example:****

**City:** Tokyo  
**Country:** Japan  
**Coordinates:** 35,6769, 139,7639  
**Time zone:** Asia/Tokyo

### ****Screenshot Placeholder:****



## ****7. Challenges Faced****

* Managing errors when the API returned no data for small towns.
* Keeping the API key private and secure using PHP.
* Ensuring the webpage stayed responsive on mobile and desktop.
* Handling asynchronous fetch operations smoothly.

## ****8. Conclusion****

The **City Explorer Web App** successfully demonstrates how to integrate an external API into a dynamic, interactive web application.  
It shows how modern web technologies (HTML, CSS, JS, PHP) can work together to fetch and display real-time data without refreshing the page.

Through this lab, the concepts of **API integration**, **asynchronous JavaScript**, and **responsive web design** were reinforced.

## ****9. Future Enhancements****

* Integrate a **weather API** to show the city’s current weather.
* Add **Teleport API** support to display quality-of-life and population data.
* Cache previous searches to improve performance.
* Provide a dark mode UI theme.

## ****10. References****

* OpenCage API Documentation – https://opencagedata.com/api
* Bootstrap 5 Documentation – <https://getbootstrap.com/>
* MDN Web Docs (Fetch API) – https://developer.mozilla.org/en-US/docs/Web/API/Fetch\_API
* OpenStreetMap – https://www.openstreetmap.org