# **Application Setup Guide**

## **Overview**

This guide provides step-by-step instructions to set up and run the application. The project comprises two main components:

1. **BackEnd**: Developed using Python with the FastAPI framework.
2. **WebApp**: Built with JavaScript using Vite and React.

## **Prerequisites**

### Database

The backend project utilizes MySQL, and you'll need to configure the following information in the backend's **.env** file:

* **DATABASE\_HOSTNAME**
* **DATABASE\_PORT**
* **DATABASE\_PASSWORD**
* **DATABASE\_NAME**
* **DATABASE\_USERNAME**

Create the database manually, and ensure that the specified database name (**DATABASE\_NAME**) matches the created schema.

## **Backend Setup**

1. Open a terminal and navigate to the **Backend** folder:

1. cd BackEnd

1. Install the required Python packages:

1. pip install -r requirements.txt

1. Launch the backend application:

1. uvicorn app.main:app --reload

Note: The initial launch may take a minute as it performs web scraping to retrieve locations.

1. Once the backend is running, send a GET request to the following endpoint to create an API key:

1. GET http://{replace with your ip}:8000/API/createAPI/

Example:

1. GET http://127.0.0.1:8000/API/createAPI/

Note: You can use tool such as Postman or just visit the link in you web chrome.

1. Copy the generated API key for the Web App Setup.

## **Web App Setup**

1. Navigate to the **Web App** folder:

1. cd WebApp

1. Install the required npm packages:

1. npm install

1. Open the **.env** file in the **Web App** folder and paste the copied API key:

1. REACT\_APP\_API\_KEY=YOUR\_COPIED\_API\_KEY

1. Save the file.
2. Run the web app:

1. npm run dev

Now, your application is set up and running. Visit the specified URL (typically **http://localhost:5173**) to view the web app with markers on the retrieved coordinates.

# **Documentation of Key Logic and Ideas in Getting the Location**

## **Overview**

**LocationScraper** uses an iterative approach with the OpenStreetMap API to find locations for shops when full addresses are not accurately recognized by the free service.

## **Key Strategies**

1. **Combining Address Components**:
   * Due to OpenStreetMap's limited accuracy for full addresses, the script uses combinations of address components:
     1. **Shop Name + Jalan + Postal Code**: Most descriptive, tried first.
     2. **Shop Name + Postal Code**: When 'jalan' is unclear or absent.
     3. **Jalan + Postal Code**: Final attempt, excluding the shop name.
2. **Regex for Address Parsing**:
   * Regular expressions are employed to parse 'jalan' and postal codes, focusing on elements that OpenStreetMap can more reliably interpret.
3. **Error Handling**:
   * Errors are logged for troubleshooting, important for refining the geolocation process.

## **Process**

* Iterates over each shop, creating search parameters from address components.
* Queries OpenStreetMap API with these parameters.
* Stops at the first successful geolocation result for each shop.

## **Usage**

* Effective when dealing with addresses where free geocoding services like OpenStreetMap lack precision.
* This method is a practical solution for scenarios where a more accurate, paid geolocation service is not feasible.
* Particularly useful in contexts where addresses are non-standard, incomplete, or the accuracy requirement is limited to the vicinity of the shop name, street, and postal code.

## **Conclusion**

**LocationScraper** is a pragmatic approach to geolocation, optimizing address components for the best results with free resources.