Setting Up a Python Server and Resolving Dynamic Linking Issues

When developing a web project locally, testing files directly in a browser can cause issues due to security restrictions. This guide explains why a Python server is necessary for dynamically fetching files like JSON and HTML components, and how to set it up properly.

Why Use a Python Server?

1. **CORS Restrictions**
   Modern browsers enforce Cross-Origin Resource Sharing (CORS) policies. When loading files locally (`file://` protocol), browsers block dynamic requests for JSON data or external HTML imports.

2. **Dynamic Content Loading**
   Projects like *Badger-Icons* dynamically load categories, subcategories, and images from a JSON file. A server is required to facilitate these requests properly.

3. **Simulating a Live Environment**
   Running a local server mimics a real hosting setup, ensuring the website behaves as expected before deployment.

Setting Up a Python Server

Step 1: Navigate to the Project Directory

Open a terminal or command prompt and move to the folder containing `index.html` , `header.html` , and `directory-structure.json` :

```
cd /path/to/project
```

Step 2: Start the Python Server

For Python 3, use the following command:

```
python -m http.server 8000
```

This starts a lightweight HTTP server on port **8000**. Your project will be accessible at:

```
http://localhost:8000
```

Step 3: Verify the Setup

Open a browser and visit `http://localhost:8000` . Your site should load correctly, and dynamic files should be accessible.

Dynamically Fetching Files

Example: `index.html`

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Badger-Icons</title>
  </head>
  <body>
    <section class="container mt-5">
      <div class="row">
        <!-- Dynamic content will be inserted here -->
      </div>
    </section>
  </body>
</html>
```

Example: `index.js` (For GitHub Pages Deployment)

```javascript
fetch("https://madhurimarawat.github.io/Badger-Icons/directory-structure.json")
  .then((response) => {
    if (!response.ok) {
      throw new Error(`HTTP error! Status: ${response.status}`);
    }
    return response.json();
  })
  .then((data) => console.log("Data loaded from GitHub Pages:", data))
  .catch((error) => console.error("Error loading JSON:", error));
```

Example: `index-local.js` (For Local Development)

```javascript
fetch("directory-structure.json")
  .then((response) => {
    if (!response.ok) {
      throw new Error(`HTTP error! Status: ${response.status}`);
    }
    return response.json();
  })
  .then((data) => console.log("Data loaded locally:", data))
  .catch((error) => console.error("Error loading JSON:", error));
```

Troubleshooting Common Issues

1. **Incorrect File Paths**
   Ensure that files are correctly placed relative to `index.html`. Verify the paths used in `fetch()` requests.

2. **Browser Console Errors**
   Open Developer Tools (`F12` in most browsers) and check for errors in the console.

3. **Port Conflicts**
   If port **8000** is in use, start the server on a different port:

   ```
   python -m http.server 8080
   ```

   Then access your site at `http://localhost:8080`.

By running a Python server, *Badger-Icons* and similar projects can function seamlessly both locally and when deployed, ensuring a smooth development workflow.