🚀 React and Vite App Deployment with GitHub Pages

✅ Issue Description

```
GET https://username.github.io/src/main.jsx net::ERR_ABORTED 404
```

What this means:

- The `index.html` file is attempting to load `src/main.jsx` directly from the server.
- On **production (GitHub Pages)**, it should reference the **bundled and built files**, rather than the raw `.jsx` files.
- This error typically indicates that the **source code** has been deployed instead of the **compiled files** from the `dist` folder.

✅ How to Fix It (Step-by-Step Guide)

Step 1: Install `gh-pages`

The developer should install the `gh-pages` package as a development dependency. Inside the project root directory, they can run:

```
npm install --save-dev gh-pages
```

Step 2: Update `vite.config.js`

In the `vite.config.js` file, ensure the `base` property correctly points to the GitHub repository name. This is critical for GitHub Pages to resolve paths correctly.

```
import { defineConfig } from "vite";
import react from "@vitejs/plugin-react";

export default defineConfig({
  base: "/repository-name/", // 👈 Replace with the repository name!
```

```
  plugins: [react()],
});
```

Step 3: Edit `package.json` Scripts

In the `package.json` file, add or update the scripts section to include deploy commands. This automates the build and deploy process.

```
"scripts": {
  "dev": "vite",
  "build": "vite build",
  "preview": "vite preview",
  "predeploy": "npm run build",
  "deploy": "gh-pages -d dist"
}
```

✅ Link the Local Project to GitHub Repository

Step 4: Set the Correct Remote Repository

The developer should verify and set the GitHub remote URL using:

```
git remote set-url origin https://github.com/username/repository-name.git
```

They can confirm it with:

```
git remote -v
```

Expected output:

```
origin  https://github.com/username/repository-name.git (fetch)
origin  https://github.com/username/repository-name.git (push)
```

Step 5: Commit and Push the Code

If there are changes in the local project, the developer should commit and push them to GitHub using:

```
git add .
git commit -m "Updated site for gh-pages deployment"
git push origin main
```

(Replace `main` with the appropriate branch name if necessary)

✅ Deploy to GitHub Pages

Step 6: Ensure Correct Deployment Settings

a) Inside `package.json`:

Make sure the `deploy` script explicitly points to the repository:

```
"deploy": "gh-pages -d dist -r https://github.com/username/repository-name.git"
```

b) Or run the deploy command manually:

```
npx gh-pages -d dist -r https://github.com/username/repository-name.git
```

Step 7: Deploy the Project

The developer can now deploy the project by running:

```
npm run deploy
```

✅ What happens during deployment:

- The app is built into the `/dist` folder.
- The contents of `dist` are pushed to the `gh-pages` branch.
- GitHub Pages serves the **built files**, not the raw source files.

✅ Configure GitHub Pages Settings

Step 8: Select Deployment Source in GitHub

1. Go to the repository:
   👉 `https://github.com/username/repository-name`
2. Navigate to:
   Settings → **Pages**
3. Under **Source**:
   - Select **Branch**: `gh-pages`
   - Select **Folder**: `/ (root)`

Step 9: Access the Live Site

After ~1 minute, the deployed website should be live at:
👉 `https://username.github.io/repository-name/`

✅ Recap of the Deployment Process

🔹 What was wrong?

The project was referencing `src/main.jsx`, which doesn't exist in the production build. GitHub Pages serves from `/dist/index.html` and requires compiled, bundled files.

🔹 What was done?

- The app was built using `vite build`.
- The `/dist` folder was deployed with `gh-pages`.

✅ What Not to Do

❌ Don'ts
❌ Don't deploy raw `src` files to GitHub Pages
❌ Don't manually upload files to GitHub Pages
❌ Don't serve `main.jsx` —use the **built** files!

✅ Quick Commands Summary

Action	Command
Install	`npm install gh-pages --save-dev`
Build	`npm run build`
Build & Deploy	`npm run deploy`
GitHub Pages	Configure GitHub Pages to use the `gh-pages` branch

✅ Important Note for Future Deployments

For every new deployment, the developer should follow these steps to ensure the latest production build is published:

1. **Run the Deploy Command**

   ```
   npm run deploy
   ```

   ✅ This command **automatically runs the build process first**, because the `predeploy` script in `package.json` triggers `npm run build` before deploying. It builds the project and pushes the latest `/dist` files to the `gh-pages` branch.

🔹 When to Use `npm run build` Separately?

Running `npm run build` on its own is only necessary if the developer wants to:

- **Test the production build locally**
  For example, by running:

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

- **Inspect the contents of the `/dist` folder**
- **Prepare files for manual deployment or uploading elsewhere**

✅ Quick Recap

- For standard deployments:
  ➡️ Just run `npm run deploy` .

- For testing the build before deploying (optional):
  ➡️ Run `npm run build` followed by `npm run preview`.

✅ Final Result

👉 After following all steps correctly, the website will be successfully live at:
`https://username.github.io/repository-name/`