STATIC SITE GENERATION

Lab Part 1: Introduction to Static Site Generators

Objective:

Understand the basics of Static Site Generators and their role in modern web development. Learn to set up an SSG environment using a popular framework (e.g., Next.js).

Tools Required:

- Node.js (for Next.js)
- Git and GitHub (for version control)
- Visual Studio Code (or any code editor)
- Terminal/Command Prompt

Steps:

1. **Introduction to SSG:**
    - Understand what SSG is and its benefits (e.g., speed, security, simplicity) as explained in last lab.
    - Research popular SSG frameworks (Next.js, Hugo, Gatsby).
2. **Setup:**
    - Install Node.js from nodejs.org.
    - Install Next.js using the following commands:

      ```
      npm init -y
      ```

      ```
      npm install next react-dom
      ```
    - Create a pages directory and add a basic index.js:

      ```
      export default function Home() {
        return <h1>Hello, Static Site Generation!</h1>;
      }
      ```
3. **Run the Project:**
    - Add a script to package.json:

```
"scripts": {
  "dev": "next dev"
}
```

- Start the development server:

```
npm run dev
```

4. **Generate a Static Site:**
   - Add a build script:

     ```
     "scripts": {
       "build": "next build && next export"
     }
     ```

   - Run the build command:

     ```
     npm run build
     ```

   - Check the out folder for static HTML files.

Lab Part 2: Creating Dynamic Pages with Markdown

Objective:

Learn how to use Markdown files for content and generate dynamic pages using an SSG framework.

Steps:

1. **Install Markdown Dependencies:**

   ```
   npm install gray-matter remark remark-html
   ```

2. **Create a Markdown File:**
   - Create a posts directory and add hello-world.md:

     ```
     ---
     title: "Hello World"
     ```

```
date: "2024-11-16"
---
```

This is a Markdown file for our static site.

3. **Write a Function to Parse Markdown:**
    - Add a utility function to read and parse the Markdown file:

```
import fs from 'fs';
import path from 'path';
import matter from 'gray-matter';

export function getPostData() {
 const filePath = path.join(process.cwd(), 'posts', 'hello-world.md');
 const fileContents = fs.readFileSync(filePath, 'utf8');
 const { data, content } = matter(fileContents);
 return { data, content };
}
```

4. **Render the Markdown Content:**
    - Update index.js to display the Markdown content:

```
import { getPostData } from '../lib/posts';

export default function Home() {
 const { data, content } = getPostData();
 return (
  <div>
   <h1>{data.title}</h1>
   <p>{data.date}</p>
   <div dangerouslySetInnerHTML={{ __html: content }} />
```

```
    </div>
  );
}
```

5. **Build and Export the Static Site:**

   ```
   npm run build
   ```

Lab Part 3: Deploying a Static Site Using GitHub Pages

Objective:

Deploy the static site generated in the previous lab to GitHub Pages.

Prerequisites:

- GitHub account
- GitHub repository

Steps:

1. **Initialize Git and Push to GitHub:**

   ```
   git init
   git add .
   git commit -m "Initial commit"
   git branch -M main
   git remote add origin <your-github-repo-url>
   git push -u origin main
   ```

2. **Configure GitHub Pages:**
   - Go to the repository settings on GitHub.
   - Enable GitHub Pages from the gh-pages branch (you may need to set up a GitHub Action to automatically deploy).
3. **Deploy the Site:**
   - Install gh-pages:

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

- Add the deploy script to package.json:

```
"scripts": {
  "deploy": "next build && next export && gh-pages -d out"
}
```

- Run the deploy command:

```
npm run deploy
```

4. **Verify Deployment:**
   - Visit your GitHub Pages URL to see the deployed site (e.g., https://your-username.github.io/your-repo/).

Lab Part 4: Adding SEO and Meta Tags

Objective:

Learn how to enhance your static site with SEO best practices and meta tags.

Steps:

1. **Install Next.js Head Component:**
   - Use the <Head> component to add meta tags in index.js:

```
import Head from 'next/head';


export default function Home() {
  return (
    <>
      <Head>
        <title>My Static Site</title>
        <meta name="description" content="This is a sample static site." />
```

```
      <meta property="og:title" content="My Static Site" />
      <meta property="og:description" content="This is a sample static site using Next.js." />
    </Head>
    <h1>Welcome to My Static Site</h1>
   </>
 );
}
```

2. **Build and Verify:**

   ```
   npm run build
   ```

3. **Check the HTML Output:**
   - Inspect the index.html file in the out folder to verify that the meta tags are correctly included.

Lab Part 5: Optimizing for Performance and Accessibility

Objective:

Optimize the static site for performance and accessibility using tools like Lighthouse.

Steps:

1. **Run Lighthouse Audit:**
   - Open your site in Chrome, open DevTools, and go to the Lighthouse tab.
   - Click "Generate Report" and review the performance, accessibility, and SEO scores.
2. **Implement Performance Improvements:**
   - Use **lazy loading** for images:

   ```
   <img src="/image.png" alt="Example Image" loading="lazy" />
   ```

3. **Check Accessibility:**
   - Ensure all images have alt attributes.

- Use semantic HTML tags (<header>, <main>, <footer>).

Lab Manual: Static Site Generation Using Gatsby & GraphQl

Lab Part 1: Introduction to Gatsby and Setup

Objective:

Learn the basics of Static Site Generation and set up a Gatsby project.

Tools Required:

- Node.js (Download from nodejs.org)
- Git and GitHub (for version control)
- Visual Studio Code (or any code editor)
- Terminal/Command Prompt

Steps:

1. **Introduction to Gatsby:**
    - Review the benefits of using Gatsby for Static Site Generation (e.g., speed, security, scalability).
    - Understand how Gatsby uses **React**, **GraphQL**, and **plugins** to build modern static websites.
2. **Installing Gatsby CLI:**
    - Install the Gatsby CLI globally:

    npm install -g gatsby-cli
3. **Creating a New Gatsby Project:**
    - Create a new Gatsby site using the default starter template:

    gatsby new gatsby-site

```
cd gatsby-site
```

4. **Running the Development Server:**
    - Start the Gatsby development server:

```
gatsby develop
```

    - Open http://localhost:8000 to view your site.
5. **Understanding the Project Structure:**
    - Discuss key directories:
        - src: Contains the source code (components, pages, styles).
        - public: Holds the built static files.
        - gatsby-config.js: Configuration file for plugins and site metadata.

Lab Part 2: Creating Pages and Using GraphQL

Objective:

Learn how to create pages in Gatsby and use GraphQL for data querying.

Steps:

1. **Creating a New Page:**
    - Create a new page called about.js in the src/pages directory:

```
// src/pages/about.js
import React from "react";


const AboutPage = () => (
 <main>
  <h1>About Us</h1>
  <p>This is the about page of our Gatsby site.</p>
 </main>
);
```

```
export default AboutPage;
```

2. **Adding Navigation:**
   - Update the index.js file to include a link to the About page:

```
import React from "react";
import { Link } from "gatsby";

const HomePage = () => (
 <main>
  <h1>Welcome to Gatsby</h1>
  <Link to="/about">Go to About Page</Link>
 </main>
);

export default HomePage;
```

3. **Using GraphQL in Gatsby:**
   - Open the GraphQL playground at http://localhost:8000/___graphql and explore data querying.
   - Example query:

```
{
 site {
  siteMetadata {
   title
   description
  }
 }
```

```
}
```

- Update gatsby-config.js with site metadata:

```
module.exports = {
  siteMetadata: {
    title: "My Gatsby Site",
    description: "A simple site built with Gatsby",
  },
};
```

4. **Displaying Metadata:**
   - Use GraphQL in index.js to fetch and display the site title:

```
import React from "react";
import { graphql } from "gatsby";

const HomePage = ({ data }) => (
  <main>
    <h1>{data.site.siteMetadata.title}</h1>
    <p>{data.site.siteMetadata.description}</p>
  </main>
);

export const query = graphql`
  query {
    site {
      siteMetadata {
        title
        description
```

```
    }
  }
}
`;

export default HomePage;
```

Lab Part 3: Using Markdown for Content

Objective:

Learn how to use Markdown files for content and generate dynamic pages using Gatsby's filesystem plugin.

Steps:

1. **Install the Filesystem and Transformer Plugins:**

   ```
   npm install gatsby-source-filesystem gatsby-transformer-remark
   ```

2. **Configure Plugins in gatsby-config.js:**

   ```
   module.exports = {
    plugins: [
     {
      resolve: "gatsby-source-filesystem",
      options: {
       name: "posts",
       path: `${__dirname}/src/posts/`,
      },
     },
     "gatsby-transformer-remark",
    ],
   ```

```
};
```

3. **Create a Markdown File:**
    - Add hello-world.md in the src/posts/ directory:

```
---
title: "Hello World"
date: "2024-11-16"
---
```

This is a Markdown file for our Gatsby site.

4. **Create a Template to Render Markdown:**

```
import React from "react";

const BlogPost = ({ data }) => {
 const post = data.markdownRemark;
 return (
  <main>
   <h1>{post.frontmatter.title}</h1>
   <p>{post.frontmatter.date}</p>
   <div dangerouslySetInnerHTML={{ __html: post.html }} />
  </main>
 );
};

export const query = graphql`
 query($slug: String!) {
  markdownRemark(fields: { slug: { eq: $slug } }) {
   html
```

```
    frontmatter {
      title
      date
    }
  }
}
`;

export default BlogPost;
```

5. **Run the Build Command:**

```
gatsby build
```

Lab Part 4: Deploying to GitHub Pages

Objective:

Learn how to deploy a Gatsby site to GitHub Pages.

Steps:

1. **Install GitHub Pages Plugin:**

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

2. **Update gatsby-config.js for GitHub Pages:**

```
module.exports = {
  pathPrefix: "/your-repo-name",
};
```

3. **Add Deploy Script:**

```
"scripts": {
  "deploy": "gatsby build && gh-pages -d public"
```

```
}
```

4. **Deploy the Site:**

```
npm run deploy
```

LabPart 5: Optimizing for Performance and SEO

Objective:

Learn how to enhance the performance and SEO of a Gatsby site.

Steps:

1. **Install SEO Plugin:**

```
npm install gatsby-plugin-react-helmet react-helmet
```

2. **Configure Plugin:**

```
module.exports = {
 plugins: ["gatsby-plugin-react-helmet"],
};
```

3. **Add Meta Tags Using React Helmet:**

```
import React from "react";
import { Helmet } from "react-helmet";

const SEO = () => (
 <Helmet>
  <title>My Gatsby Site</title>
  <meta name="description" content="This is a Gatsby site optimized for SEO" />
 </Helmet>
);

export default SEO;
```