Project Documentation

Project Overview

File Types Found:

- javascript: 14 files

- config: 3 files

- web: 4 files

- documentation: 1 files

File: eslint.config.js (javascript)

1. File Overview: This is an ESLint configuration file that defines the settings and rules for a React project using the eslint-plugin-react, eslint-plugin-react-hooks, and eslint-plugin-react-refresh plugins. The file exports an array of objects, each representing a separate set of configurations for ESLint.

2. Key Components:

* `import js from "@eslint/js";`: This line imports the base ESLint configuration from the `@eslint/js` package.

* `import globals from "globals";`: This line imports the globals module, which provides a list of predefined global variables for the project.

* `import react from "eslint-plugin-react";`: This line imports the eslint-plugin-react plugin, which provides additional rules and configurations for React projects.

* `import reactHooks from "eslint-plugin-react-hooks";`: This line imports the eslint-plugin-react-hooks plugin, which provides additional rules and configurations for React hooks.

* `import reactRefresh from "eslint-plugin-react-refresh";`: This line imports the

eslint-plugin-react-refresh plugin, which provides additional rules and configurations for React Refresh.

3. Dependencies:

* `@eslint/js`: This is the base ESLint configuration that defines the default rules and settings for all projects.

* `globals`: This module provides a list of predefined global variables for the project.

* `eslint-plugin-react`: This plugin provides additional rules and configurations for React projects.

* `eslint-plugin-react-hooks`: This plugin provides additional rules and configurations for React hooks.

* `eslint-plugin-react-refresh`: This plugin provides additional rules and configurations for React Refresh.

4. Implementation Details:

* The file exports an array of objects, each representing a separate set of configurations for ESLint.

* Each object in the array defines a set of configuration options for ESLint, including the `ignores` option, which specifies files or directories to ignore during linting, and the `files`, `languageOptions`, `settings`, `plugins`, and `rules` options, which define the specific rules and settings for ESLint.

* The `js` module is imported to provide the base ESLint configuration.

* The `globals` module is imported to provide a list of predefined global variables for the project.

* The `react`, `react-hooks`, and `react-refresh` plugins are imported to provide additional rules and configurations for React projects, hooks, and Refresh.

* The `rules` option defines the specific rules and settings for ESLint, including the `jsx-runtime` rule from the react plugin, which specifies the preferred JSX runtime to use in the project.

5. Usage Examples:

To use this configuration file with ESLint, you would need to run the following command in your terminal or command prompt:

```bash
```

eslint --config eslint.config.js path/to/your/project/directory

```

This will lint all JavaScript and JSX files in the `path/to/your/project/directory` directory using the configuration defined in the file.

## File: firebase.json (config)

File Overview:

The `firebase.json` file is a configuration file for Firebase Hosting, which allows developers to define the behavior of their website or application when it's deployed on Firebase Hosting. This file specifies the public directory, the files and directories that should be ignored, rewrites for URLs, and other settings.

Key Components:

1. `hosting` - The main component in this file is the `hosting` section, which contains all the configuration options for Firebase Hosting.
2. `public` - This property specifies the directory where the public files are located. In this case, it's set to `"dist"`.
3. `ignore` - This property specifies the files and directories that should be ignored by Firebase Hosting. It includes the `firebase.json` file itself, as well as any files or directories starting with a dot (`"**/.*"`), and any subdirectories within the `node_modules` directory (`"**/node_modules/**"`).
4. `rewrites` - This property specifies how URLs should be rewritten by Firebase Hosting. In this case, it's set to a single rewrite rule that matches any URL (`"**"`) and redirects it to the `/index.html` file.
```

Dependencies:

This file does not require any external dependencies or modules. It is standalone and can be used as is.

Implementation Details:

The `firebase.json` file uses a simple syntax for configuration, with each property being defined within its own section. The `rewrites` property is particularly interesting, as it demonstrates the use of regex to match any URL (`"**"`). This allows for easy and flexible URL rewriting.

Usage Examples:

This file can be used as part of a Firebase Hosting deployment process, where it defines the behavior of the website or application when it's deployed on Firebase Hosting. It can also be used as a reference for developers who want to understand how to configure their Firebase Hosting deployment.

File: index.html (web)

1. File Overview: This HTML file is the main entry point for a web application, providing a basic structure and layout for the website. It includes the basic metadata such as title, description, keywords, author, and open graph tags for social media sharing.

2. Key Components:

* Head tag: This section contains the basic metadata and links to external stylesheets, scripts, and fonts.

* Body tag: This section contains the main content of the web page, including a div with an ID of "root" that will be used by the JavaScript code later in the file.

3. Dependencies:

* src/main.jsx: This is the main entry point for the JavaScript code, which is included in the body tag using a script tag with the type="module" attribute. The contents of this file are not part of the HTML documentation and are therefore not covered here.

4. Implementation Details:

* HTML structure: The HTML document has a basic structure of head and body tags, with some additional metadata and links to external assets in the head tag.

* SEO Meta Tags: The meta tags used in this file provide information about the web page's title, description, keywords, author, and open graph image for social media sharing.

5. Usage Examples:

This HTML file is a basic structure that can be modified to create a more complex web application. It includes some basic metadata and links to external assets such as fonts and scripts, but the actual content of the page is not included here.

File: package.json (config)

File Overview:

The package.json file is a configuration file for a Node.js project that uses the Vite bundler and React library. It specifies the dependencies, scripts, and other metadata required to manage the project's development and build processes.

Key Components:

* "name": The name of the project, which is "portfolio" in this case.

* "private": A boolean value indicating whether the project should be published as a private package. In this case, the project is marked as private.

* "version": The version number of the project. This value is not used for publishing purposes.

* "type": The type of package, which is set to "module" in this case. This indicates that the project

uses ECMAScript modules and can be imported using the import statement.

* "scripts": A dictionary of scripts that can be run from the command line. The dev script runs the Vite server, the build script builds the project, the lint script checks for code errors, and the preview script previews the built project.

* "dependencies": A list of dependencies required by the project. In this case, the project depends on React, React-dom, Tailwind CSS, and Vite.

* "devDependencies": A list of development dependencies that are not included in the published package but are used during development. The project depends on eslint, eslint-plugin-react, eslint-plugin-react-hooks, eslint-plugin-react-refresh, globals, and vite.

Dependencies:

* React: A popular JavaScript library for building user interfaces.

* React-dom: The DOM (Document Object Model) implementation of React that allows rendering React components in the browser.

* Tailwind CSS: A utility-first CSS framework that simplifies styling and design customization.

* Vite: A fast, configurable build tool for modern web applications. It provides a lightweight alternative to other popular bundlers like Webpack or Rollup.

Implementation Details:

* The project uses the React library for building user interfaces.

* The project depends on Tailwind CSS for styling and design customization.

* The project uses Vite as a build tool, which provides a lightweight alternative to other popular bundlers like Webpack or Rollup.

Usage Examples:

1. To start the development server, run the following command in the terminal:

npm run dev

2. To build the project, run the following command in the terminal:

   npm run build

3. To check for code errors, run the following command in the terminal:

   npm run lint

4. To preview the built project, run the following command in the terminal:

   npm run preview

File: README.md (documentation)

1. File Overview:

This file provides an overview of the React + Vite template, which is a minimal setup to get React working in Vite with HMR and some ESLint rules. The purpose of this file is to provide a brief description of the file's content and its intended use case.

2. Key Components:

The main classes, functions, or structures in this file are:

* @vitejs/plugin-react: This plugin uses Babel for Fast Refresh.

* @vitejs/plugin-react-swc: This plugin uses SWC for Fast Refresh.

* ESLint rules: These are the recommended ESLint rules for React projects using Vite.

3. Dependencies:

This file depends on several external libraries and modules, including:

* Vite: A tool for building and bundling modern web applications.

* React: A JavaScript library for building user interfaces.

* Babel: A JavaScript compiler that transpiles code from one version of JavaScript to another.

* SWC: A Rust-based JavaScript compiler that transpiles code from one version of JavaScript to another.

* ESLint: A tool for identifying and reporting on patterns in JavaScript code.

4. Implementation Details:

The implementation details of this file include the use of Fast Refresh, which allows developers to quickly update their code without having to restart the development server. This is achieved by using a plugin that provides HMR support for React components. Additionally, the file includes some recommended ESLint rules for React projects using Vite.

5. Usage Examples:

The usage examples of this file include setting up a new React project using Vite and adding Fast Refresh support using one of the available plugins. For example, developers can run the command "vite init my-react-app" to create a new React project and then install the required dependencies, including Vite, React, Babel, and ESLint. Once the dependencies are installed, developers can start the development server by running the command "npm start". Whenever they make changes to their code, Fast Refresh will automatically update the application without requiring a restart of the development server.

File: vite.config.js (javascript)

1. File Overview: This file is a configuration file for Vite, which is a web application builder and toolchain designed by Evan You. It is used to configure the development environment and build process for Vite applications. The file imports the `defineConfig` function from the "vite" module, as well as two plugins: `react` and `tailwindcss`.

2. Key Components:

* `defineConfig`: This function is used to define the configuration for a Vite application. It takes an object with various properties that configure the build process. In this file, it is called with an empty object as its argument, which means that no additional configuration is provided.

* `react` and `tailwindcss`: These are plugins for Vite that provide functionality related to React and Tailwind CSS. The `react` plugin allows you to use React components in your Vite application,

while the `tailwindcss` plugin enables you to use Tailwind CSS in your application.

3. Dependencies:

* "vite": This is a dependency that provides the core functionality for Vite. It includes the `defineConfig` function and other features that are used in building and running Vite applications.

* "@vitejs/plugin-react": This is a dependency that provides support for using React components in your Vite application.

* "@tailwindcss/vite": This is a dependency that provides support for using Tailwind CSS in your Vite application.

4. Implementation Details:

* The `defineConfig` function is used to define the configuration for a Vite application. It takes an object with various properties that configure the build process. In this file, it is called with an empty object as its argument, which means that no additional configuration is provided.

* The `react` and `tailwindcss` plugins are used to provide functionality related to React and Tailwind CSS, respectively. They can be added to the `plugins` property of the `defineConfig` function to enable their respective features.

5. Usage Examples:

* To use this configuration file in a Vite application, you would import it at the top of your entry point file and call the `defineConfig` function with an empty object as its argument. For example:

```
import viteConfig from "./vite.config";

export default defineConfig({});
```

This will configure the Vite application to use the React and Tailwind CSS plugins, but no additional configuration is provided. You can then use the `vite` command in your terminal to build and run the application.

File: public\404.html (web)

1. File Overview: The public/404.html file is a customizable error page that displays when a user visits a URL that does not exist on the website. It provides a simple, consistent layout for all 404 errors and allows users to easily understand what went wrong and how they can fix it.

2. Key Components: The key components of this file include:

* Doctype declaration (<!DOCTYPE html>)
* HTML element (<html>)
* Head element (<head>)
* Meta element (<meta charset="utf-8">) and (<meta name="viewport" content="width=device-width, initial-scale=1">)
* Title element (<title>Page Not Found</title>)
* Body element (<body>)
* Div element ( )
* H2 element (<h2>404</h2>)
* H1 element (<h1>Page Not Found</h1>)
* P element (The specified file was not found on this website. Please check the URL for mistakes and try again.)
* H3 element (<h3>Why am I seeing this?</h3>)
* P element (This page was generated by the Firebase Command-Line Interface. To modify it, edit the <code>404.html</code> file in your project's configured <code>public</code> directory.)

3. Dependencies: This file does not require any external dependencies or libraries. It is a standalone HTML file that can be viewed by anyone with access to it.

4. Implementation Details: The implementation details of this file include the use of CSS styles and JavaScript functions to create a consistent layout for all 404 errors, as well as the use of semantic HTML elements to provide structure and meaning to the page. The file also uses inline styling using

the style attribute in HTML tags.

5. Usage Examples: This file can be used to customize the error pages generated by a Firebase project's server. It can be edited to change the text, images, or layout of the error pages to better match the needs of the user. Additionally, it can be used as a starting point for creating new error pages with different designs and layouts.

File: public\index.html (web)

File Overview:

This HTML file is part of a web project that uses Firebase Hosting to deploy a web application. The file sets up the Firebase SDK and initializes it with the necessary features, such as authentication, real-time database, Firestore, and storage. It also includes some code for logging events and errors using Firebase Analytics and Performance Monitoring.

Key Components:

1. Firebase SDK initialization script: This script loads the Firebase JavaScript library and initializes it with the necessary features. The `firebase.app()` object is used to access the initialized Firebase SDK instance, which provides access to various features such as authentication, real-time database, Firestore, and storage.

2. Firebase Analytics and Performance Monitoring: These are two separate modules provided by Firebase that allow developers to track events and monitor performance metrics of their web applications. The code snippet for logging events is included in the file, but it is commented out.

3. DOMContentLoaded event listener: This script attaches an event listener to the `DOMContentLoaded` event, which is fired when the HTML document has finished loading and parsing. Once the event is triggered, the initialization script is executed.

Note: The code in this file is only for initializing Firebase SDK with necessary features. It is not intended to be a comprehensive tutorial on how to use Firebase Hosting or any other Firebase services.

File: src\App.jsx (javascript)

File Overview: This is a JavaScript file for an application that uses React. The file contains the main class, `App`, which returns a component with various components and layout elements. It also imports several other files, including components like `Header`, `Hero`, `Skills`, `Projects`, `Contact`, and `Footer`.

Key Components:

* `App`: This is the main class that defines the root component of the application.
* `Header`: This is a component that displays the header section of the page, which typically contains the application's name, logo, and navigation menu.
* `Hero`: This is a component that displays the hero section of the page, which typically contains an image or video background, a heading, and a subheading.
* `Skills`: This is a component that displays the skills section of the page, which typically lists the application's key features and technologies used.
* `Projects`: This is a component that displays the projects section of the page, which typically contains a list of projects or portfolios created by the application.
* `Contact`: This is a component that displays the contact section of the page, which typically contains a form for users to send messages or inquiries to the application's developers or support staff.
* `Footer`: This is a component that displays the footer section of the page, which typically contains copyright information and links to other pages or resources.

Dependencies:

* React: This is an essential dependency for building web applications using React. It provides the necessary tools and libraries for creating, rendering, and managing user interfaces.
* Other external dependencies: Depending on the specific requirements of the application, other libraries and modules may be required to interact with APIs, databases, or other systems.

Implementation Details:

* Importing and exporting components: This file uses React's component hierarchy to organize the different components into a tree-like structure. Components are imported using ES6 `import` syntax, and they are exported using the `export default` statement.
* Using CSS classes and styles: This file uses CSS classes and styles to customize the appearance of the application's components. The `bg-gray-900 text-gray-100 min-h-screen` class is used to set the background color, text color, and minimum height of the component.
* Using JSX syntax: This file uses JSX syntax to create dynamic HTML elements in the React render method. The `<>` symbol is used to define a fragment, which contains multiple components or elements that are not wrapped in a parent element.

Usage Examples:

* To use this file as a standalone application, developers would need to import and export the necessary dependencies and create an instance of the `App` class using the `ReactDOM.render()` method. For example:

```
import React from "react";
import { render } from "react-dom";
import App from "./src/App";
```

```
const rootElement = document.getElementById("root");
render(<App />, rootElement);
```

* To use this file as a module in another project, developers would need to import the necessary dependencies and create an instance of the `App` class using the `ReactDOM.render()` method within their own application's render method. For example:

```
import React from "react";
import { render } from "react-dom";
import App from "./src/App";

const rootElement = document.getElementById("root");
render(<App />, rootElement);
```

This file is a fundamental part of any web application built using React, as it provides the basic structure and layout for the entire application.

File: src\index.css (web)

File Overview:

This is a CSS file used for styling web pages. The purpose of the file is to define the basic styles and layout for the web page, using TailwindCSS as the main framework.

Key Components:

* `@import "tailwindcss";`: This line imports the TailwindCSS library, which provides a pre-defined set of CSS classes that can be used to style elements on the page.
* `root { ... }`: This section defines the root element of the HTML document, which is typically the `<html>` tag. The styles defined in this block apply to all elements within the root element.
* `font-family: Inter, system-ui, Avenir, Helvetica, Arial, sans-serif;`: This line sets the default font family for the page to "Inter", and falls back to "system-ui" if "Inter" is not available. The font "Avenir" is also included as a fallback option, followed by "Helvetica" and "Arial". Finally, "sans-serif" is used as a catch-all fallback for any browser that does not support the previous fonts.
* `line-height: 1.5;`: This line sets the default line height for all elements on the page to 1.5 times the font size.
* `font-weight: 400;`: This line sets the default font weight for all elements on the page to 400 (normal).
* `font-synthesis: none;`: This line disables the automatic synthesis of fonts, which can help improve performance and reduce CPU usage.
* `text-rendering: optimizeLegibility;`: This line sets the default text rendering mode for all elements on the page to "optimize legibility", which prioritizes legibility over rendering speed.
* `-webkit-font-smoothing: antialiased;`: This line enables anti-aliasing for fonts in WebKit browsers, which can help improve the appearance of fonts and reduce blurriness.
* `-moz-osx-font-smoothing: grayscale;`: This line enables grayscaling for fonts in Mozilla Firefox on macOS, which can help improve the appearance of fonts and reduce blurriness.

Dependencies:

This CSS file does not have any external dependencies beyond TailwindCSS itself.

Implementation Details:

TailwindCSS is a popular CSS framework that provides a pre-defined set of classes for styling

HTML elements. This CSS file uses the `@import` statement to import the TailwindCSS library and then defines styles for the root element using the `root { ... }` block. The font family, line height, and font weight are all defined here, as well as some other important styles such as text rendering mode and anti-aliasing.

Usage Examples:

This CSS file can be used to style any HTML page by simply including it in the `<head>` section of the document using a `<link>` tag. For example:

```html
<head>
  <link rel="stylesheet" href="src/index.css">
</head>
```

This will import the styles defined in this file and apply them to all elements within the HTML document.

File: src\main.jsx (javascript)

File Overview: This is the main entry point of the application, where the React framework and the rest of the project's dependencies are imported. The file creates a root element in the DOM using `createRoot` from `react-dom/client`, which serves as the container for the entire application. It then renders the `App` component within the `StrictMode` context, which helps to identify and fix any problems with the code that might be caused by side effects or unexpected behavior.

Key Components:

* `StrictMode`: A React feature that helps to catch potential bugs in the code by forcing components

to use the latest versions of their dependencies and helping to identify problems related to stale state.

* `createRoot`: A function from the `react-dom/client` package that creates a root element in the DOM and returns an object with methods for rendering React components.
* `App`: The main component of the application, which is imported from the `./App.jsx` file.

Dependencies:

* `react`: The core React library used for building user interfaces.
* `react-dom/client`: A package that provides a way to render React components in the browser using JavaScript in the main thread (i.e., not inside web workers).
* `./index.css`: An external stylesheet file imported by the application, which contains CSS rules for styling the UI elements.

Implementation Details:

* The `createRoot` function is used to create a root element in the DOM and return an object with methods for rendering React components.
* The `StrictMode` component is used to wrap the `App` component within it, which helps to identify and fix any problems with the code that might be caused by side effects or unexpected behavior.
* The `render` method of the root element returned by `createRoot` is used to render the `App` component within the `StrictMode` context.

Usage Examples:

* To run the application, simply import the `./main.jsx` file in the browser or using a tool like

Webpack. The `App` component will be rendered in the `#root` element of the DOM.

File: .github\workflows\firebase-hosting-deploy.yml (config)

File Overview:

This file is a GitHub Actions workflow that deploys a web app to Firebase Hosting on each merge to the `main` branch. The file uses the `firebase-hosting-deploy.yml` template provided by the Firebase Extended GitHub Action.

Key Components:

* `name`: The name of the workflow, which is "Deploy to Firebase Hosting on merge".
* `on`: The event that triggers the workflow, which is a push to the `main` branch.
* `permissions`: The permissions required for the workflow to run, which is write access to the repository's contents.
* `jobs`: The job that runs when the workflow is triggered, which is "build_and_deploy".
* `runs-on`: The operating system used by the job, which is Ubuntu latest.
* `steps`: A list of steps that make up the job, including:
  + `uses`: The Firebase Extended GitHub Action for deploying to Firebase Hosting.
  + `with`: The configuration options for the action, including:
    - `repoToken`: The repository token used for authentication with Firebase.
    - `firebaseServiceAccount`: The service account credentials for Firebase.
    - `channelId`: The ID of the channel to deploy to, which is either "live" or an empty string depending on whether the push event is a merge or not.
    - `projectId`: The ID of the Firebase project to deploy to, which is "beekeeper-f2377".

Dependencies:

* `actions/checkout@v4`
* `actions/setup-node@v3`
* `FirebaseExtended/action-hosting-deploy@v0`

Implementation Details:
The workflow uses the Firebase Extended GitHub Action to deploy the web app to Firebase Hosting. The action takes in several configuration options, including the repository token, service account credentials, channel ID, and project ID. The workflow checks if the push event is a merge and sets the `channelId` accordingly. If the push event is not a merge, the `channelId` is set to an empty string.

Usage Examples:
The workflow is triggered on each push to the `main` branch, which will deploy the web app to Firebase Hosting using the configured service account credentials and channel ID. The workflow can be customized by modifying the `with` section of the `uses` step to include additional configuration options or different deployment parameters.

File: src\components\Contact\Contact.jsx (javascript)

File Overview: This is a React component file for a contact form. It uses the FadeInSection component from the Utilities directory to add a fade-in effect to the section when it is first rendered. The contact form consists of a text input field, an email input field, a textarea for the message, and a submit button.

Key Components:

1. Contact - This is the main component that defines the contact form layout and functionality. It uses the FadeInSection component to add a fade-in effect to the section when it is first rendered.
2. FadeInSection - This is a custom component that adds a fade-in effect to the section when it is first rendered. It takes in a child component as its only prop and wraps it in a div with a fade-in animation.

Dependencies:

1. React - This is the core library for building user interfaces in React.
2. FadeInSection - This is a custom component that adds a fade-in effect to the section when it is first rendered. It takes in a child component as its only prop and wraps it in a div with a fade-in animation.

Implementation Details:

1. The Contact component uses the FadeInSection component to add a fade-in effect to the section when it is first rendered. This allows for a smooth transition from the initial load of the page to the contact form being displayed.
2. The contact form consists of a text input field, an email input field, a textarea for the message, and a submit button. Each of these fields has custom CSS classes applied to them to style them with a dark gray background and white text. The textarea has a height of 32 pixels to accommodate longer messages.
3. When the form is submitted, it uses the HTML5 "form" attribute to validate the input and submit the data to the server. The form also includes a CSS class called "transition-colors duration-300"

which adds a transition animation to the button when it is hovered over or clicked.

Usage Examples:

1. This component can be used in any React project that requires a contact form with a fade-in effect. It can be imported and used directly in other components, or it can be used as a base for creating more complex contact forms.

File: src\components\Experience\Experience.jsx (javascript)

1. File Overview: This file is a React component for displaying the user's professional experience on a website. It consists of a section with an ID of "experience" that displays a list of companies and their associated jobs, as well as a brief description of each job. The file also imports the FadeInSection component from another module.
2. Key Components:
* `Experience`: This is the main class for this component, which defines the structure and behavior of the experience section on the website. It includes a list of companies and their associated jobs, as well as a brief description of each job.
* `FadeInSection`: This is a separate component that is imported from another module. It is used to create a fade-in effect for the experience section, making it appear more visually appealing.
3. Dependencies: The file depends on the React library and the FadeInSection component from another module.
4. Implementation Details: The implementation of this component involves using React's JSX syntax to define the structure of the component, as well as leveraging CSS styles to style the experience section. The component also uses the `FadeInSection` component to create a fade-in effect for the experience section.

5. Usage Examples: This component can be used to display a user's professional experience on a website, highlighting their relevant work experience and skills. It can be imported and rendered in another component to add it to the website. For example:

```
import React from "react";
import Experience from "../components/Experience/Experience";

const MyComponent = () => (
  <div>
    <Experience />
  </div>
);

export default MyComponent;
```

File: src\components\Footer\Footer.jsx (javascript)

File Overview:

The `src\components\Footer\Footer.jsx` file is a React component that renders a footer with a copyright notice and an optional link to the author's website.

Key Components:

* `React`: This is the main class used in this file, which is responsible for creating the React element.
* `Footer`: This is the functional component that renders the footer. It takes no props and returns a

JSX element with a footer class.

* `footerClassName`: This is a string that specifies the CSS class name for the footer. In this case, it is set to "bg-gray-900 text-gray-400 py-8".
* `max-w-7xl mx-auto px-4`: These are CSS classes that specify the maximum width of the container, margin, and padding for the footer.
* `text-center`: This is a CSS class that centers the text within the footer.
* `© 2024 Arundhati Das. All rights reserved.`: This is the copyright notice that appears in the footer.
* `Optional link to author's website`: This is an optional link to the author's website, which can be added as a hyperlink within the footer.

Dependencies:

This file does not have any external dependencies. It only uses the React library to create the component.

Implementation Details:

The implementation of this file uses React functional components and JSX syntax to create the component. The `Footer` component takes no props and returns a JSX element with a footer class. The CSS classes used in this file are set using className properties, which are assigned using strings. The `max-w-7xl mx-auto px-4` classes are used to specify the maximum width of the container, margin, and padding for the footer, while the `text-center` class is used to center the text within the footer.

Usage Examples:

This file can be imported into other React components to use the `Footer` component. For example:

```
import Footer from "../components/Footer/Footer";
```

```
function MyComponent() {
  return (
    <div>
      <h1>My Component</h1>
      <Footer />
    </div>
  );
}
```

File: src\components\Header\Header.jsx (javascript)

File Overview:

This is a React component file that defines a header navigation bar for the AD website. The header includes a logo, a list of links to different sections of the website, and a hamburger menu button for mobile devices. The header is a fixed element at the top of the screen and has a transparent background with a blur effect.

Key Components:

1. `Header` component: This is the main component that defines the structure and functionality of the header navigation bar. It includes a logo, a list of links to different sections of the website, and a hamburger menu button for mobile devices. The `Header` component uses the `useState` hook from React to manage the state of the hamburger menu button.
2. `NavLink` component: This is a child component of the `Header` component that defines each link

in the header navigation bar. It includes an `onClick` event handler that scrolls the user to the corresponding section of the website when clicked.

3. `useState` hook: This is a React hook used to manage the state of the hamburger menu button. It is used to set and update the `isOpen` state variable, which determines whether the hamburger menu is open or closed.

4. `onClick` event handler: This is an event handler that is attached to each link in the header navigation bar. It scrolls the user to the corresponding section of the website when clicked.

5. `document.getElementById(sectionId)?.scrollIntoView({ behavior: "smooth" })`: This is a JavaScript function used to scroll the user to the corresponding section of the website when the link is clicked. The `sectionId` parameter is passed as an argument and is used to identify the corresponding section of the website.

Dependencies:

This file depends on the React library, which provides the functionality for building dynamic UI components. It also depends on the NavLink component from the same project, which defines each link in the header navigation bar.

Implementation Details:

The `Header` component uses a combination of CSS classes and JavaScript functions to create a responsive and interactive header navigation bar. The hamburger menu button is hidden on desktop devices and only visible on mobile devices. When clicked, it opens or closes the hamburger menu, which displays a list of links to different sections of the website.

Usage Examples:

This file can be imported into other React components to use the `Header` component. For example, it can be used in a layout component to add a header navigation bar to the page.

File: src\components\Hero\Hero.jsx (javascript)

File Overview: This is a React component file for the Hero section of a website. It defines a `Hero` component that displays a hero image, header text, and a button to scroll to other sections of the page. The `Hero` component uses the `TypewriterText` component to display animated typewriter text with a purple gradient background.

Key Components:

* `Hero`: The main component of this file that defines the Hero section of the website. It has several key components such as the hero image, header text, and button to scroll to other sections.
* `TypewriterText`: A separate component that displays animated typewriter text with a purple gradient background.
* `DynamicBackground`: A component that provides a dynamic background for the Hero section by using the `useEffect` hook to update the background image every 10 seconds.

Dependencies:

* React
* useState and useEffect from React's Hooks API
* DynamicBackground from src/Utilities/DynamicBackground.jsx

Implementation Details:

* The `Hero` component uses a combination of CSS classes and inline styles to create a responsive design that works well on both desktop and mobile devices.

* The `TypewriterText` component uses the `useState` and `useEffect` hooks to manage its state and update it every 100 milliseconds, which is equivalent to one character per second. It also has an animation class called "animate-fadeIn" that fades in the text over a period of one second.
* The `DynamicBackground` component uses the `useEffect` hook to update the background image every 10 seconds, which provides a dynamic and changing background for the Hero section.

Usage Examples:

* To use the `Hero` component in another React file, you can import it and render it like any other component. For example:

```
import { Hero } from "./src/components/Hero/Hero";

function App() {
  return (
    <div>
      <Hero />
    </div>
  );
}
```

* To use the `TypewriterText` component in another React file, you can import it and render it like any other component. For example:

```
import { TypewriterText } from "./src/components/Hero/TypewriterText";
```

```
function App() {
  return (
    <div>
      <TypewriterText />
    </div>
  );
}
```

File: src\components\NavLink\NavLink.jsx (javascript)

1. File Overview: This file defines a React component for creating navigation links with a purple color when hovered over.

2. Key Components:

* `NavLink`: The main class of the component, which is exported as the default export from the file. It is a functional component that takes two props: `onClick` and `children`.

* `button`: A HTML button element that is used to create the navigation link.

3. Dependencies:

* React: This file requires the React library to be imported in order to use its components and functionality.

4. Implementation Details:

* The `NavLink` component uses a transition effect to change the color of the text when it is hovered over, which is achieved by using the `transition-colors` class on the button element. This effect requires the user's browser to support CSS transitions, but most modern browsers do.

5. Usage Examples:

```jsx
import React from 'react';
```

```
import NavLink from './NavLink';

function MyComponent() {
  return (
    <div>
      <NavLink onClick={() => console.log('clicked!')}>Click me</NavLink>
    </div>
  );
}
```

In this example, the `MyComponent` function uses the `NavLink` component to create a navigation link with the text "Click me". When the user clicks on the link, the `onClick` prop is called, which logs the message "clicked!" to the console.

File: src\components\Project\Project.jsx (javascript)

File Overview: This file is a React component that displays a list of projects with their details. The component uses the FadeInSection utility to create a fade-in animation on the project cards.

Key Components:

1. `React` - A JavaScript library for building user interfaces.
2. `FadeInSection` - An external utility component that creates a fade-in animation on the project cards.
3. `Projects` - The main class of the component, which displays the list of projects with their details.

Dependencies:

1. `React` - A JavaScript library for building user interfaces.
2. `FadeInSection` - An external utility component that creates a fade-in animation on the project cards.

Implementation Details:

1. The component uses the `map()` method to iterate over the list of projects and create a new array of project components. Each project component is rendered with a fade-in animation using the `FadeInSection` utility.
2. The component uses the `grid` class from Tailwind CSS to create a responsive grid layout for the project cards. The grid has 3 columns on desktop and 1 column on mobile.
3. The component uses the `text-purple-400` class from Tailwind CSS to apply a custom text color to the project titles and descriptions.
4. The component uses the `bg-gray-800/30 rounded-lg p-8 backdrop-blur-sm transform hover:scale-105 transition-all duration-300` class from Tailwind CSS to apply a custom background color, border radius, padding, and animation to the project cards.

Usage Examples:

1. The component can be used in any React component that requires a list of projects with their details. For example:

```
import React from 'react';
import Projects from './Projects';
```

```
function App() {
  return (
    <div className="container mx-auto">
      <Projects />
    </div>
  );
}

export default App;
```

2. The component can be customized using props, such as the `title`, `description`, and `technologies` properties of each project. For example:

```
import React from 'react';
import Projects from './Projects';

function App() {
  return (
    <div className="container mx-auto">
      <Projects
        projects={[
          {
            title: "GPU Architecture Benchmarking",
            description: "Advanced performance profiling for RTX 30/40 Series GPUs",
            technologies: ["Python", "NSight", "CUDA"],
```

```
        },
        {
          title: "Weather Data Dashboard",
          description: "Real-time weather monitoring with 10-minute updates",
          technologies: ["AWS Lambda", "QuickSight", "Athena"],
        },
      ]}
    />
  </div>
 );
}

export default App;
```

File: src\components\Skills\Skills.jsx (javascript)

File Overview:

This file contains a React component called `Skills` that displays a list of skills with categories and subcategories. The component uses the `FadeInSection` component from the `Utilities` directory to provide an animation effect when rendering the content.

Key Components:

* `Skills`: This is the main component that renders the skill categories and subcategories. It uses the `map()` method to iterate over an array of objects with `title` and `skills` properties, and renders a

separate `FadeInSection` component for each category.

* `FadeInSection`: This is a React component that provides an animation effect when rendering its child components. It uses the `transition-all duration-500` CSS class to animate the opacity of the child components, and the `hover:scale-105 hover:bg-gray-800/70` CSS classes to change the background color and scale of the child components on hover.

Dependencies:

* React: This is a JavaScript library used for building user interfaces. It is imported at the top of the file using the `import React from "react";` statement.
* FadeInSection: This is a custom component that provides an animation effect when rendering its child components. It is imported from the `Utilities` directory using the `import FadeInSection from "../Utilities/FadeInSection";` statement.

Implementation Details:

* The `Skills` component uses the `map()` method to iterate over an array of objects with `title` and `skills` properties, and renders a separate `FadeInSection` component for each category. Each `FadeInSection` component contains a heading (`h3`) with the category title, followed by a list (`ul`) of subcategories (`li`).
* The `FadeInSection` component uses the `transition-all duration-500` CSS class to animate the opacity of its child components. When hovered over, it changes the background color and scale of its child components using the `hover:scale-105 hover:bg-gray-800/70` CSS classes.
* The `Skills` component also uses the `bg-gray-800/50 rounded-lg transform transition-all duration-500 hover:scale-105 hover:bg-gray-800/70` CSS class to animate the background color and scale of its child components when hovered over.

Usage Examples:

* To use this component, simply import it into a React file using the `import Skills from "src/components/Skills";` statement and render it within your application's layout or template. For example: `<div><Skills /></div>`.

File: src\components\Utilities\DynamicBackground.jsx (javascript)

File Overview:

The `DynamicBackground` component is a React functional component that generates and animates particles on the screen. It uses the `useState` hook to store an array of particle objects, and the `useEffect` hook to update their positions and render them in the background. The component also includes a radial gradient as a background layer with a blur effect.

Key Components:

* `particles`: An array of particle objects with properties such as position (`x`, `y`), size, speed, and opacity.
* `generateParticles()`: A function that generates an initial set of particles at the page's dimensions.
* `updateParticles()`: A function that updates the positions of the particles and re-renders them in the background. It is called repeatedly using the `setInterval` function with a delay of 16 milliseconds.
* `<div className="absolute inset-0 z-0 overflow-hidden">`: An HTML div element that contains the particle animation and sets its position, size, and z-index.

* `<svg>`: An SVG element that is used to render the particles as circles.
* `bg-gradient-to-br from-gray-900 via-purple-900/20 to-gray-900`: A CSS gradient background layer with a purple color transition at the bottom of the screen.
* `bg-[radial-gradient(circle_at_50%_50%,rgba(147,51,234,0.1),transparent_70%)]`: A CSS radial gradient background layer with a gray color transition at the bottom of the screen.
* `backdrop-blur-[1px]`: A CSS blur effect on the parent element to create a hazy effect.

Dependencies:

* React
* Radial gradient
* Blur effect

Implementation Details:

* The `useState` hook is used to store an array of particle objects, and the `useEffect` hook is used to update their positions and render them in the background.
* The `generateParticles()` function generates an initial set of particles at the page's dimensions using the `Array.from` method and a random number generator.
* The `updateParticles()` function updates the positions of the particles and re-renders them in the background using the `setInterval` function with a delay of 16 milliseconds. It also uses the `map` method to create a new array of particle objects with updated positions.
* The component uses a radial gradient background layer with a purple color transition at the bottom of the screen, and a blur effect on the parent element to create a hazy effect.

Usage Examples:

* To use the `DynamicBackground` component in another React component, import it and use it as follows:

```javascript
import React from 'react';
import DynamicBackground from './components/Utilities/DynamicBackground';

function MyComponent() {
  return (
    <div className="my-component">
      <DynamicBackground />
      {/* Other content */}
    </div>
  );
}
```

* To adjust the number of particles or their speed, modify the `length` parameter in the `generateParticles()` function and the `speedX` and `speedY` properties in the particle object.

File: src\components\Utilities\FadeInSection.jsx (javascript)

File Overview:

This file contains a React component named `FadeInSection` that allows for fading in elements based on their visibility in the viewport. The component takes an optional `delay` prop, which defaults to 0, and uses it to control the delay of the transition.

Key Components:

* `useState`: This is a React hook used to manage state. It creates a state variable that can be updated with the `setVisible` function, and returns the current value of the variable along with a function to update it. In this case, the state variable `isVisible` is used to track whether an element is visible in the viewport or not.
* `useEffect`: This is another React hook used for side effects, like setting up and cleaning up observers. It takes two arguments: a function that runs immediately after the component mounts (called the "effect" function), and an array of dependencies that determine when the effect function runs. In this case, the effect function sets up an intersection observer to track changes in visibility, and returns a cleanup function that unobserves the element when the component is unmounted.
* `IntersectionObserver`: This is a browser API used for tracking elements that are visible in the viewport. It can also be used to observe elements as they become hidden or shown. In this case, the intersection observer is set up to track changes in visibility of the element referenced by the `domRef` variable.
* `React.useRef`: This is a React hook used to create a reference object that persists across renders. It can be used to store a reference to an element or other object, and allows for updating it without causing unnecessary re-renders. In this case, the ref object stores a reference to the `div` element that wraps the child elements of the component.
* `className`: This is a prop that allows you to pass a string as a class name to the component. In this case, the className is used to apply CSS classes to the component based on its visibility in the viewport.
* `style`: This is a prop that allows you to pass an object with inline styles to the component. In this case, the style object is used to add a transition delay to the component based on the value of the `delay` prop.

Dependencies:

This file depends on React, and it also depends on the `IntersectionObserver` API provided by the browser.

Implementation Details:

The implementation of this component uses the `useState` and `useEffect` hooks to manage state and side effects, respectively. The intersection observer is set up using the `new IntersectionObserver` constructor, and it observes changes in visibility of the element referenced by the `domRef` variable. When the visibility of the element changes, the component updates its state accordingly, causing a re-render with the updated classes and styles.

Usage Examples:

Here are some usage examples for this component:

```jsx
// Basic usage
<FadeInSection>Hello world!</FadeInSection>

// With a delay
<FadeInSection delay={1000}>Hello world!</FadeInSection>

// With a custom class name
<FadeInSection className="my-class">Hello world!</FadeInSection>

// With a custom style
<FadeInSection style={{ backgroundColor: "red" }}>Hello world!</FadeInSection>
```