Mina Roshankar

Project Documentation

This project helps me provide information about myself, my portfolio and my resume to whom ever that I want to work with and removes the need to send PDF files, explain everything, and so on.

Technologies used are:

- Next.js
- React JS
- Mantine UI
- CSS
- TypeScript
- Babel
- Webpack
- TipTap
- Tailwind
- Postcss
- esLint

Architecture and infrastructure

- *Front-end Architecture:*

React Components: Next.js follows the React component-based architecture. You organize your UI into reusable components that encapsulate their own logic and rendering.
Pages: Next.js uses a file-based routing system where each file in the "pages" directory represents a route in the application. These files typically export React components that define the structure and behavior of each page.
Routing: Next.js handles client-side and server-side routing automatically based on the files in the "pages" directory. You can use dynamic routes and nested routes to build complex applications.

- *Server-side Rendering (SSR):*

Next.js provides built-in support for server-side rendering, allowing you to pre-render pages on the server before sending them to the client. SSR improves performance and enables search engine optimization (SEO) by delivering fully rendered HTML to the client.
API Routes: Next.js allows you to define serverless API endpoints using API routes. These routes can handle server-side logic, data fetching, and interact with databases or external APIs.

- *Static Site Generation (SSG):*

Next.js supports static site generation, where pages are pre-rendered at build time and served as static files. This approach is ideal for content-driven websites or pages that don't require real-time data.
Data Fetching: Next.js provides a data fetching framework that allows you to fetch data at build time and pre-render pages with that data. You can use server-side functions or external APIs to fetch data during the build process.
Infrastructure:

- *Hosting:*

Next.js applications can be hosted on various platforms, including traditional web servers, cloud platforms (such as AWS, Azure, or Google Cloud), or serverless platforms (such as Vercel, Netlify, or AWS Lambda).

- *Build Process:*

Next.js applications require a build step where the application is compiled and optimized for production. This process generates static HTML files, JavaScript bundles, and other assets.
Deployment: Next.js applications can be deployed using different deployment strategies, such as continuous integration and continuous deployment (CI/CD) pipelines, Git-based deployments, or manual deployments via command-line interfaces.
Additional Considerations:

- *Styling:*

Next.js supports various styling options, including CSS modules, CSS-in-JS libraries like styled-components or Emotion, or using global CSS files.
State Management: Next.js does not enforce a specific state management solution. You can use popular libraries like Redux, MobX, or React Context API to manage application state.
Testing: Next.js applications can be tested using standard JavaScript testing frameworks like Jest or React Testing Library.

Deployment on the web

Deploying a Next.js application on the web using Cloudflare Pages is a straightforward process that allows you to host your application with ease. Here's a simple description of how you can deploy a Next.js application on Cloudflare Pages:

Sign up for Cloudflare Pages: Visit the Cloudflare Pages website (https://pages.cloudflare.com/) and sign up for an account if you don't already have one. Cloudflare Pages offers a free plan that you can use for most small-scale projects.

Create a new project: Once you're signed in, click on the "Create a project" button to start setting up your Next.js application. Provide a name for your project and select the GitHub repository or Git repository where your Next.js code is stored.

Configure the build settings: Cloudflare Pages automatically detects that you're using Next.js and sets up the build configuration for you. However, you can customize the build command and output directory if needed. By default, the build command is set to "npm run build" and the output directory is set to "out".

Set up the environment variables: If your Next.js application requires environment variables, you can set them up in the "Environment Variables" section of your project settings. Add any necessary variables and their values.

Build and deploy the application: Once your project is configured, Cloudflare Pages will automatically trigger a build process based on your repository's default branch. It will run the build command specified in the settings and generate the optimized production files.

Review and test the deployment: Once the build process is complete, Cloudflare Pages will provide you with a preview URL where you can review and test your deployed Next.js application. Verify that everything looks and functions as expected.

Configure custom domain (optional): If you want to use a custom domain for your Next.js application, Cloudflare Pages makes it easy to set up. In your project settings, navigate to the "Custom Domains" section and follow the instructions to add and configure your custom domain.

Deploy updates: Whenever you push updates to your repository, Cloudflare Pages will automatically rebuild and redeploy your Next.js application, ensuring that the latest changes are reflected on your live site.

Cloudflare Pages takes care of the infrastructure and hosting for your Next.js application, providing a scalable and globally distributed network for serving your content. This allows you to focus on building your application while Cloudflare handles the deployment and delivery process.

Major components

Front-end Components:

Pages: Next.js uses a file-based routing system where each file in the "pages" directory represents a route in the application. These files typically export React components that define the structure and behavior of each page.
React Components: Next.js follows the React component-based architecture. You organize your UI into reusable components that encapsulate their own logic and rendering.
Routing: Next.js handles client-side routing automatically based on the files in the "pages" directory. You can use dynamic routes and nested routes to build complex applications.

State Management:

React Context API: Next.js provides built-in support for React's Context API, which allows you to manage and share global state across components without having to pass props explicitly.
External State Management Libraries: You can also utilize popular state management libraries like Redux, MobX, or Zustand to handle complex state management requirements in your Next.js application.
Backend Integration:

API Routes: Next.js allows you to define serverless API endpoints using API routes. These routes can handle server-side logic, data fetching, and interact with databases or external APIs.
Data Fetching: Next.js provides several methods for fetching data in a Next.js application. You can use the built-in getStaticProps or getServerSideProps functions to fetch data at build time or request time, respectively. Alternatively, you can use client-side data fetching libraries like axios or fetch to fetch data directly from the client-side.
These components work together to create a dynamic and interactive SPA built with Next.js. The front-end components define the UI structure and behavior, while state management allows you to manage and share data across components. Backend integration enables you to interact with APIs and fetch data to populate your application with dynamic content.

It's important to note that Next.js provides a flexible architecture, allowing you to choose the specific tools and libraries that best fit your project's needs. The components mentioned above represent the core elements of a Next.js SPA, but the specific implementation can vary depending on the requirements of your application.