Day 4 - Technical Report : MORENT

INTRODUCTION

This technical report documents the development process of a modern web application built using Next.js, Tailwind CSS, and TypeScript. The project integrates essential components, API connections, and user authentication while following industry best practices to ensure performance, scalability, and maintainability.

1. Steps Taken to Build and Integrate

Planning and Design:

- Defined the project objectives and feature requirements.
- Created wireframes and UI/UX mockups for a clear visual guide.

Setting up the Development Environment:

- Initialized a Next.js project using create-next-app.
- Installed dependencies, including Tailwind CSS for styling and TypeScript for type safety.
- Configured the project structure with a src directory for better organization.

Component Development:

- Built reusable UI components (e.g., buttons, search bar, cards) to maintain consistency.
- Implemented a Layout component to provide a common structure across pages.

API Integration:

- Connected to a public API to fetch weather data using Server-Side Rendering (SSR) for dynamic data and Static Site Generation (SSG) for performance optimization.
- Utilized getServerSideProps and getStaticProps functions to handle API requests.

Authentication:

- Implemented basic user authentication using a session-based approach.
- Created login and registration components with secure password storage.

2. Challenges Faced and Solutions

Data Feeding Efficiency:

- **Challenge:** Delays and performance issues during data fetching.
- **Solution:** Used SSR for frequently changing data and SSG for static content to optimize performance.

Search Bar Integration:

- **Challenge:** Handling dynamic suggestions and user input.
- **Solution:** Implemented debounce techniques to limit API calls and improved routing with useRouter.

Authentication Security:

- **Challenge:** Securing user data during login and registration.
- **Solution:** Applied hashing algorithms for passwords to enhance security.

3. Best Practices Followed

Component Reusability:

- Followed the DRY (Don't Repeat Yourself) principle by creating modular, reusable components.

Code Readability:

- Maintained clean and readable code with proper comments and descriptive variable names.

Type Safety:

- Leveraged TypeScript for improved type safety and fewer runtime errors.

Performance Optimization:

- Used SSR and SSG appropriately to enhance application speed and user experience.