**Dynamic Personal Portfolio Website**

**Project Overview**

The project is a dynamic, single-page portfolio website meticulously crafted to serve as a central hub for my professional profile. It effectively showcases my skills, development experience, completed projects, and certifications in a clean, modern, and interactive format. The website is structured into distinct, navigable sections: **About**, **Experience**, **What I Do**, **Skills**, **Projects**, **Internships**, **Certificates**, and **Contact Me**.

**Design Choices & Technology Stack**

The core philosophy was to build a highly performant and visually appealing website using foundational web technologies to demonstrate strong core development skills.

* **Structure: HTML5**
  + **Reasoning:** The site is built on a foundation of semantic HTML5. Using tags like <nav>, <section>, and <footer> ensures the website is well-structured, accessible for screen readers, and optimized for search engines (SEO).
* **Styling: CSS3**
  + **Reasoning:** All styling was done with custom CSS3 to achieve a unique and polished look. Key features include a sophisticated **dark theme**, a custom-built **vertical timeline** for the experience section, and responsive card-based layouts. Modern CSS modules like **Flexbox** and **Grid** were used extensively to create a fluid and responsive design that adapts perfectly across all devices, from mobile phones to desktops.
* **Interactivity: JavaScript (Vanilla JS)**
  + **Reasoning:** To keep the site lightweight and demonstrate mastery of core programming concepts, all interactivity was implemented using plain JavaScript. This avoids the overhead of external frameworks. The JavaScript code handles all the dynamic features of the site.
* **Deployment: Netlify**
  + **Reasoning:** The website is deployed on Netlify, as seen in the URL. Netlify was chosen for its seamless Git-based workflow, which allows for **continuous deployment**—automatically updating the live site whenever changes are pushed to the GitHub repository. It also provides a global CDN for fast load times and handles backend functionalities like form submissions.

**Key Features & Implementation**

* **Smooth Scrolling Navigation**
  + **Implementation:** The fixed navigation bar at the top allows users to jump to any section. This was achieved by adding a JavaScript click event listener to each navigation link. When a link is clicked, event.preventDefault() stops the default jump, and the window.scrollTo() method is used with { behavior: 'smooth' } to create a fluid scrolling animation to the target section.
* **Active Link Highlighting on Scroll**
  + **Implementation:** A key feature is that the navigation link corresponding to the section currently in the viewport is automatically highlighted. This was implemented by adding a scroll event listener to the window. The handler function calculates the scroll position and uses getBoundingClientRect() for each section to determine which one is active, then adds an active-link CSS class to the correct navigation item.
* **Dynamic Experience Timeline**
  + **Implementation:** The "Experience" and "Internships" sections feature a visually engaging vertical timeline. This was constructed using CSS pseudo-elements (::before and ::after). A central line was created using a pseudo-element on the timeline container, while the circles marking each event were created as pseudo-elements on the individual timeline items, showcasing advanced CSS capabilities.
* **Responsive Project & Certificate Grids**
  + **Implementation:** The "Projects" and "Certificates" sections use a responsive grid system (likely CSS Grid) to display multiple cards. This allows the gallery to cleanly adjust its column count based on the screen width, ensuring the layout is always organized and visually balanced.

**Challenges & Solutions**

1. **Challenge: Ensuring Performant Scrolling with Active Highlighting**
   * **Problem:** Attaching complex calculations directly to the scroll event can cause performance issues, as the event fires very frequently.
   * **Solution:** To optimize this, the scroll handling function was **throttled**. Throttling ensures that the position-checking logic only runs once every few milliseconds (e.g., 100ms) instead of on every single scroll pixel, which significantly improves performance while maintaining a smooth user experience.
2. **Challenge: Optimizing Load Time with Many Images**
   * **Problem:** The certificates section contains numerous high-resolution images, which could drastically increase the initial load time of the website.
   * **Solution:** I implemented **lazy loading** for all images that are not immediately visible in the viewport. By adding the loading="lazy" attribute to the <img> tags, the browser delays loading these images until the user scrolls them into view. This simple attribute dramatically improves the initial page load speed.