1. I have experience in web development, particularly in areas like front-end development, back-end development, and full-stack development.

I. Front-end Development:

a. HTML/CSS: I Use html and css for creating the structure and styling of web pages.

b. JavaScript: I Use for adding interactivity and dynamic behavior to web pages. (Bringing Life to the static web pages I build with html and css)

c. Front-end frameworks/libraries: Such as React.js, Next.js for building interactive user interfaces.

d. Responsive design: Using frameworks like Bootstrap, Material UI breakpoints to ensure websites work well on various devices.

II. Back-end Development:

1. Server-side languages: Like Node.js (with Express.js), for handling server-side logic.
2. Databases: SQL databases like PostgreSQL, as well as NoSQL databases like MongoDB.
3. RESTful APIs: Designing and consuming APIs for communication between the front-end and back-end.

III. Full-stack Development:

I combine front-end and back-end technologies to develop complete web applications. I understand the entire development process, from designing user interfaces to implementing server-side logic and database management.

IV. Version Control:

Git and GitHub:

I Use for version control and collaboration with other developers. (e.g. To push and pull codes )

V. Development Tools:

I use Integrated Development Environments (IDEs) such as Visual Studio Code for writing and managing code.

I also use Command Line Interface (CLI) for tasks like project setup, running servers, and version control operations.

VI. Other Skills:

I Understand some web performance optimization techniques.

I have some Knowledge of web security best practices.

I am Familiar with design principles and UX/UI concepts for creating user-friendly interfaces.

1. I have worked in a team of developers on a Health care management system as part of the frontend team, my job description was building UIs and connecting to the backend Apis. I used Next.js for building interactive web pages and material ui for styling.
2. a. Identify the Problem Clearly:
3. I start by understanding exactly what's wrong. Check if it is a visual bug, a functional issue, or a performance problem?
4. I read the logs in browser's developer console to gather details like, when and where does it occur? Can I modify it etc.
5. If it is a complex issue, I isolate the code to review it carefully before I paste it to chatgpt or a senior developer to get details on it and help.
6. Fix, test and repeat.
7. a. Static Code Analysis:
   1. I employ static analysis tools to examine my code structure and identify potential errors or inefficiencies.
   2. This helps maintain clean, consistent, and well-formatted code.

b. Human Evaluation:

1. While I can perform automated testing, human evaluation plays a crucial role.
2. Human testers can identify issues beyond the scope of automated tests, like usability or relevance.

5. a. Continuous Learning: I am constantly ingesting and processing new information from various sources, including code repositories, research papers, technical documentation, and online discussions about web development.

b. Focus on Emerging Trends: My training algorithms are designed to identify patterns and trends in the data I consume. This allows me to stay current on the latest advancements and innovations in web development.

c. Human Expertise Integration: While I can learn independently, human experts play a crucial role in guiding my development and ensuring the accuracy and relevance of the information I provide. Human feedback helps me refine my understanding of web development trends and identify areas for further exploration.

6. One of the challenges I encountered while developing a restaurant app was persisting a countdown timer for food availability. The timer was initially emitted from the server, and I wanted it to begin counting down as soon as the admin updated a dish's status to "awaiting." However, the problem arose when refreshing the browser caused the timer to restart. To address this, I researched online and consulted ChatGPT, which provided valuable insights. I learned about using local storage to persist the wait time and leveraging the useEffect hook to fetch it upon browser refresh. This approach allowed me to calculate the elapsed time accurately, even after a page refresh.

7.

1. Prioritization and Focus: "I prioritize tasks and use focused work sessions to efficiently manage multiple projects."
2. Time Management Techniques: "I leverage time management techniques like Eisenhower Matrix and Limit distractions to ensure progress on all projects within deadlines."

8. In all the websites I build, I prioritize a mobile-first approach for the user interface (UI). Then, I ensure user-friendliness across all device sizes by utilizing CSS media queries, frameworks like Bootstrap or Material UI, and understanding breakpoint concepts.

9. For this I salute Next.js

SEO (Search Engine Optimization):

Server-side rendering (SSR): Next.js provides built-in SSR, ensuring that search engine crawlers can access fully-rendered HTML content, improving SEO by making content easily indexable.

Meta tags and structured data: Utilize Next.js Head component to add meta tags such as title, description, and structured data (e.g., JSON-LD) for better search engine visibility.

Dynamic routes and URL structure: Use dynamic routes in Next.js to create SEO-friendly URLs and ensure proper navigation for both users and search engines.

10. **Daily Standup Meetings:** I participate in daily standup meetings where team members discuss progress, plans for the day, and any blockers they are facing.

**Communication and Collaboration:** I respond promptly to messages from colleagues, which is crucial for effective collaboration in Agile teams. using tools like Slack or Clickup for task management and updating backlog status, showcasing my commitment to transparency and communication within the team.

**Brainstorming Sessions and Development Cycles:** I am always involved in brainstorming sessions, and development work cycles (including wireframing, design thinking, and determining user roles/user flow processes).

**Version Control Management:** I am conversant with version control management tools like Git, which is essential for managing code changes, collaborating with team members, and ensuring a stable development process.

**Problem-Solving and Adaptability**: I discuss any blockers or challenges I face with my teams and I am always ready to assist in any way I can when my team members are faced with challenges.

11. <https://hettie-ecommerce-frontend.vercel.app>:

**a.Project Planning and Design:**

 I got the user story from the client, then I got a template that matched her description and . modified to suit

 I carefully Planned the features and functionalities of the e-commerce platform.

 I Decide on data models for products, users, carts, orders, etc., using MongoDB schemata.

**b. Development with Next.js:**

 I Set up the Next.js project structure and configure routing for different pages (product listings, product details, cart, checkout, etc.).

 I Developed React components for reusable UI elements like product cards, navigation bars, and user account sections.

 I Leveraged Next.js features like server-side rendering (SSR) for SEO benefits and static site generation (SSG) for fast page load times.

 I Integrated a demo payment gateway API to simulate secure online transactions.

1. **Backend with Node.js and Express:**

* I Created a Node.js server using Express to handle API requests from the Next.js frontend.
* I Connected to your MongoDB database using a Node.js driver called Mongoose.
* I Developed API endpoints for functionalities like product data retrieval, user authentication, cart management, order processing, and payment integration.
* I Implemented security measures, user authentication and data validation to protect the application.

**D. Deployment and Maintenance:**

* I hosted my client application with Vercel because it works well with Next.js.
* Deploy your application to production and configure environment variables for sensitive data like database credentials.
* Monitor application performance and user behavior with analytics tools.
* Regularly update dependencies and libraries to ensure security and compatibility.

14. a. Maintainability:

Clean code is easy to understand and modify, both for yourself and future developers.

Clear structure, consistent naming conventions, and comments make it simple to navigate the codebase and identify specific functionalities.

b. Collaboration:

Clean code acts as a common language within a development team.

Everyone can understand the code's purpose and logic, fostering smoother collaboration and reducing misunderstandings.

This is especially important for larger projects with multiple developers working on different parts of the codebase.

c. Debugging:

Well-organized code is easier to debug.

You can isolate issues faster by following a logical structure and clear variable names.

Clean code reduces the likelihood of errors being introduced in the first place, as logic is easier to follow and potential problems are more readily identified.

d. Scalability:

As your web application grows in features and complexity, clean code makes it easier to scale.

A well-structured codebase can be modularized, allowing you to add new functionalities without affecting existing ones.

Clean code is more flexible and adaptable to future changes in requirements or technologies.

e. Performance:

While not always a direct consequence, clean code can sometimes have a positive impact on performance.

Clear and concise code can be more efficient to execute, and well-organized structures can help avoid redundant processes.

15.

a. Proactive Approach:

Modern Feature Detection: Instead of detecting browsers, leverage libraries like Modernizr that identify features a browser supports. This allows you to develop for functionalities, not specific browsers.

Validate Code: Use online tools like the W3C Markup Validation Service and CSS Validator to check your code for compliance with web standards. Valid code is more likely to render consistently across browsers.

b. Testing Strategies:

Multiple Browsers: Test your website on a variety of browsers and devices, including modern versions, older versions, and mobile browsers. Consider browser testing tools and services like BrowserStack or Sauce Labs to streamline this process.

Responsive Design: Ensure your website is responsive and adapts to different screen sizes and devices. This improves user experience and reduces compatibility issues.

c. Code-Level Solutions:

CSS Resets: Use a CSS reset stylesheet like Normalize.css to establish a baseline style across browsers, reducing inconsistencies in default browser styles.

Polyfills: For features missing in older browsers, consider using polyfills, which are Javascript libraries that provide browser-like functionality.

Graceful Degradation: Design your website with a core experience that works on all browsers. Then, progressively enhance the experience with features for modern browsers. This ensures everyone can access basic functionality.

d. Stay Updated:

Web Standards Evolution: Web technologies and standards are constantly evolving. Stay informed about the latest updates and adjust your code accordingly to maintain compatibility with modern browsers.

Community Resources: The web development community offers a wealth of resources on cross-browser compatibility. Utilize online forums, documentation, and tutorials to find solutions and best practices.

16. Git allows me to track changes made to files in a project over time. It creates a history of commits, which are snapshots of the project at different stages.

I can revert to previous commits, create branches for new features or fixes, and merge changes back into the main branch (typically master or main).

Local and Remote Repositories:

With Git, I work with both local repositories on my machine and remote repositories hosted on platforms like GitHub, and Bitbucket.

Local repositories allow me to make changes, commit them, and manage branches without affecting the main project until I'm ready to push changes to the remote repository.

Basic Git Commands:

git init: Initializes a new Git repository in the current directory.

git clone [repository URL]: Clones a remote repository to my local machine.

git add [file(s)]: Adds changes to the staging area before committing.

git commit -m "Commit message": Commits staged changes with a descriptive message.

git push origin [branch]: Pushes committed changes to the remote repository.

git pull origin [branch]: Fetches and merges changes from the remote repository to the local branch.

git branch [branch-name]: Creates a new branch.

git checkout [branch-name]: Switches to a different branch.

17. **a. Leverage Next.js Built-in Features:**

* **Static Site Generation (SSG):** Next.js excels at SSG, which pre-renders pages at build time. This delivers blazing-fast load times for content that doesn't change frequently (e.g., blog posts, product listings).
* **Server-Side Rendering (SSR):** For dynamic content or user-specific data, leverage SSR. Next.js renders pages on the server for each request, ensuring fresh data while maintaining good performance.
* **Incremental Static Regeneration (ISR):** ISR allows you to define how often a page is re-generated at build time. This is ideal for content that updates periodically (e.g., news articles).

**b. Image Optimization with next/image:**

* Next.js provides the next/image component for automatic image optimization. It automatically resizes images for different screen sizes, uses efficient formats (WebP where supported), and implements lazy loading for improved performance.

**c. Code Splitting:**

* Next.js automatically code-splits your application by routes. This ensures only the necessary code for each page gets downloaded, reducing initial page load times.

**d. Caching Mechanisms:**

* Leverage Next.js built-in caching for server-side rendered pages and API routes. This reduces server load and improves performance by serving cached content whenever possible.
* Configure browser caching headers to store static assets like images and CSS files locally on the user's browser, minimizing subsequent downloads.

**e. Bundle Optimization:**

* Utilize tools like @next/bundle-analyzer to analyze the size of your application bundles and identify areas for optimization. You can then remove unused code or explore techniques like tree-shaking to reduce bundle size.

**f. Third-Party Library Management:**

* Evaluate the necessity of third-party libraries and choose lightweight alternatives whenever possible. Consider using lazy loading for non-critical libraries to avoid impacting initial page load.

18., I have built an Ecommerce platform for a clothing brand and implemented paystack payment gateways into the websites, go through the doc for details on how I can

19. a. **Server Side**

I handle security on the server side through the use of jwt token verification, and role base access based control

I validate received payload from the client to ensure required data type, and structure

I store user refresh token in the cookies and set expiry, these ensure that the refresh token is not vunerable to cross site hacking

b. Client-side   
**Role base access based control:**

i. I validate user inputs from all forms in the client code-base

ii. I ensure that no user can access any page they don’t have access to

iii. I encrypt user private data before persisting them in the localStorage when working with redux state management

20. We had to demo for an app to some clients on a Thursday morning and I was notified on Wednesday, we worked all day for 6days to deliver the project to the client. I have gone 53 hours a week to fix all major functionalities in an ecommerce system