📘 Portfolio Website (React + Vite) – Detailed Notes / Interview Explanation

https://sarthakpatil-portfolio.netlify.app/

---

1. Project Introduction

This project is a fully responsive **personal portfolio website** developed using **React and Vite**.
The goal of the portfolio is to:

- Showcase my skills, technical stack, and personal branding
- Display my projects with demo links and source code
- Provide my contact information in a clean, professional layout
- Build a real-world production-ready React application
- Practice frontend development, responsive design, and deployment workflow

This project helped me understand component-based architecture, JSON-driven rendering, asset management in Vite, and real deployment challenges.

2. Motivation Behind Building the Project

I built this portfolio for three major reasons:

1. Placement/Job Interviews

Companies often check candidate portfolios before interviews. I wanted a clean, structured website that clearly communicates:

- who I am
- what skills I have

- what projects I have built

2. Hands-on Frontend Development

I wanted a real project that gives me experience in:

- React components
- CSS modules
- JSON-based data management
- Responsive design principles

3. Deployment & Hosting Knowledge

Learning how real websites are deployed (Netlify/Vercel) helps understand:

- build pipelines
- asset handling
- environment differences between development vs production
- debugging deployment issues

---

3. Tech Stack Selection — Why React + Vite?

React

I chose React because:

- It is widely used in the industry
- Component-based approach helps in clean UI structure

- Easy state and props management
- Smooth integration with JSON data

Vite

Vite is faster and more modern than Create-React-App:

- Instant dev server
- Fast HMR (hot module reload)
- Smaller production bundles
- Easy asset handling

Interview-ready explanation:
"I selected React + Vite because it aligns with modern frontend development practices, provides faster builds, and allows better control over project structure and optimization."

---

4. Project Architecture & Folder Structure

I organized the project to follow clean and scalable frontend architecture.

Key folders:

```
src/
  components/       # All UI components
  data/             # JSON files for skills, projects, experience
  assets/           # Images/icons used by components
  utils.js          # Utility functions like image path resolver
  vars.css          # Global CSS variables
  index.css         # Global base styles
  App.jsx           # Root component
  main.jsx          # Entry point
```

Why this structure?

- Components become reusable
- JSON files separate content from UI
- Easier to update skills/projects later without touching JSX
- Organized, scalable, and interview-friendly

---

5. JSON-Driven UI Rendering

Instead of hardcoding content directly inside components, I stored all data in JSON files:

📌 `skills.json`

Stores skill name + icon path.

📌 `projects.json`

Stores:

- project title
- imageSrc
- tech stack
- demo link
- GitHub link

📌 `history.json`

Stores experience (hidden right now, but preserved for future use).

Why use JSON?

Interview answer:
“Using JSON allows the website to be easily updatable. If I need to add a new skill or project, I only modify the JSON file — not the React code. This improves maintainability and separation of concerns.”

---

6. Component Development Workflow

I followed a reusable and modular approach.

Hero Component

- Introduces the user
- Contains my profile image and a short description
- Includes a CTA “Contact Me” button

About Component

- Explains my core strengths: Java, OOP, DSA, SQL, MERN
- Replaced default illustrations with a custom male character

Experience Component

- Initially showed dummy experience
- I hid the UI but kept the JSON for future updates
- Conditional rendering: shows section only if JSON is non-empty

Projects Component

- Dynamically generates each project card
- Uses project image, description, stack, and links

- Added custom-designed project icons (library, ecommerce, head massager)

Contact Component

- Displays email, GitHub, and LinkedIn
- Styled and optimized for mobile using custom media queries

---

7. Styling & Responsive Design Approach

I used **CSS Modules** for component-level styling because:

- They prevent class name collisions
- Make components self-contained
- Improve maintainability

Global Styling Using CSS Variables

`vars.css` contains:

- primary colors
- secondary colors
- typography
- shadow definitions

Responsive Design

I created breakpoints at:

- **830px** — for tablets

- **600px** — for mobile

Adjustments included:

- reducing font sizes
- adjusting padding
- improving spacing
- preventing long links from overflowing

Interview explanation:

“I didn’t use any UI frameworks because I wanted complete control over styling and to improve my CSS fundamentals.”

---

8. Deployment Challenges & Fixes

This was one of the most important learning parts.

❌ Issue 1 — Images not loading on Netlify

Cause: Vite needs a correct base path in production.

Fix:

```
export default defineConfig({
  plugins: [react()],
  base: './',
});
```

❌ Issue 2 — Website looks zoomed on deployment

Debugged:

- viewport

- container width
- scaling issues
- desktop vs mobile behavior

Cause:
Different viewport width on deployment → responsive scaling.

Fix:
Added precise mobile breakpoints and cleaned layout spacing.

Key takeaway:

“Deployment environments behave differently from local development. You must test UI responsiveness on real devices and hosting platforms.”

---

9. Soft Skills Learned During This Project

- **Debugging & Problem Solving**: especially asset path issues and CSS responsiveness
- **Attention to Detail**: mobile UI refinement
- **Structured Thinking**: JSON-based organization
- **Version Control Discipline**: meaningful commits, branching, pushing
- **Independent Research**: solving Vite + Netlify issues

Interview line:
“This project taught me how to think like a frontend engineer — structuring components, managing assets, optimizing responsiveness, and handling real-world deployment issues.”

---

10. Final Outcome & Future Enhancements

✔ Fully responsive professional portfolio

✔ Clean architecture and maintainable structure

✔ Easy to update (JSON-driven)

✔ Deployed and production-ready

Future Enhancements

- Add animations (Framer Motion)
- Add blog section
- Add dark/light mode toggle
- Add backend-connected contact form
- Improve accessibility features