Vibe Coding 2025 Cheat Sheet

This cheat sheet is your go-to guide for web development in 2025, covering the "vibe coding" philosophy, modern project stacks, a glossary of must-know terms, tech selection heuristics, system design patterns, accessibility, and search tips for developers.

Core Concepts

What Is "Vibe Coding"?

Vibe coding is a way of building websites and apps that puts the developer's feelings and intuition front and center. Instead of following strict rules, vibe coding is about working in a way that feels good, looks modern, and makes coding fun. Imagine coding like a musician improvises a song—using best practices, but letting creativity and flow lead the way. Vibe coding means choosing tools and styles that match the mood of your project, focusing on developer happiness, fast prototyping, and visually pleasing results. It's about mixing logic with creativity, making sure the code is not just correct, but also has a certain "vibe" or style that's easy to use and nice to look at[1][2].

Stacks by Project Type

1. Static Landing Pages

- **Frontend:** Next.js (with static export), Astro, Hugo, or Eleventy
- **Backend:** Usually none, or serverless for simple forms
- **Database:** Not needed, or lightweight (like Google Sheets via API)
- **Deployment:** Vercel, Netlify, or GitHub Pages

2. Interactive Web Apps

- **Frontend:** React (with Next.js or Remix), Vue (Nuxt), SvelteKit
- **Backend:** Node.js (Express, Fastify), Python (FastAPI), or Serverless Functions

- **Database:** PostgreSQL, MongoDB, or cloud databases (Supabase, Firebase)
- **Deployment:** Vercel, Netlify, AWS Amplify, or DigitalOcean App Platform

3. Full-Stack SaaS Products

- **Frontend:** Next.js (React), Nuxt (Vue), SvelteKit, or Remix
- **Backend:** Node.js, Go, Python (Django/FastAPI), or Serverless/Edge Functions
- **Database:** PostgreSQL (often with Prisma ORM), PlanetScale (MySQL), MongoDB Atlas
- **Deployment:** Vercel, AWS (Lambda, ECS), Google Cloud Run, or Azure Functions

4. AI-Powered Apps

- **Frontend:** Next.js, React, SvelteKit
- **Backend:** Node.js, Python (for ML), or Edge Functions
- **Database:** Vector databases (Pinecone, Weaviate), PostgreSQL
- **Deployment:** Vercel, AWS Lambda, or dedicated AI platforms

5. Enterprise or Modular Apps

- **Frontend:** Micro frontends (MCP, Module Federation)
- **Backend:** Microservices (Node.js, Go, .NET)
- **Database:** Polyglot persistence (mix of SQL/NoSQL)
- **Deployment:** Kubernetes, Docker Swarm, or cloud PaaS[3][4].

Glossary: Essential Modern Terms

Jamstack: A way to build websites using JavaScript, APIs, and Markup. It separates the frontend from the backend, making sites faster and easier to scale[5].

Serverless: Running code in the cloud without managing servers. You just write functions, and the cloud runs them when needed[6].

Edge Functions: Tiny pieces of code that run close to the user (at the "edge" of the network), making sites faster and more responsive[6].

SSR (Server-Side Rendering): The server makes the web page before sending it to the browser, so users see content faster and search engines can read it easily[7].

SSG (Static Site Generation): The site is built ahead of time as static files, making it super fast and cheap to host[7].

SPA (Single-Page Application): A web app that loads a single HTML page and updates content without refreshing the page.

ORM (Object-Relational Mapping): A tool that helps you use databases with code, turning database rows into objects in your programming language.

Microservices: Breaking an app into small, independent pieces that can be built and deployed separately.

Monolithic: Building the whole app as one big piece.

API: A way for different apps or parts of an app to talk to each other.

Headless CMS: A content management system that only manages content and lets you display it anywhere via APIs.

Component-Driven Design: Building UIs with reusable pieces (like LEGO blocks).

PWA (Progressive Web App): A website that acts like an app on your phone or computer.

Prisma: A popular ORM for Node.js and TypeScript.

PlanetScale: A scalable MySQL database platform for cloud apps[6][5].

Tech Selection Heuristics

1. Match the Stack to the Project:

Pick tools that fit your project’s needs. Static sites? Use SSG tools. Real-time apps? Go for frameworks with good state management and backend support.

2. Team Experience Matters:

Choose technologies your team knows, unless there’s a really good reason to learn something new. New tools are cool, but reliable, well-known ones are safer

for deadlines and debugging[8].

3. Focus on Performance and Scalability:

If you expect lots of users or fast growth, pick stacks that scale easily (like serverless or edge functions).

4. Budget and Maintenance:

Consider hosting costs and how easy it is to find developers for your stack. Simpler stacks are usually cheaper and easier to maintain[9].

5. "Vibe" and Developer Happiness:

Choose tools and workflows that make your team excited to build. A happy, creative team builds better products[1].

6. Use AI Wisely:

AI coding tools can speed things up, but don't rely on them for things you don't understand. Stick to one new thing at a time if you're experimenting[8].

System Design Patterns

API Design: REST vs. GraphQL

- **REST:** Simple, uses URLs for resources (like `/users`). Great for most apps, easy to cache, and well-understood.
- **GraphQL:** Lets clients ask for exactly what data they need. Good for complex apps (like dashboards), but can be harder to secure and cache.

State Management

- **Client State:** Use React/Vue/Svelte state, or tools like Redux, Zustand, or Pinia for bigger apps.
- **Server State:** Use SWR, React Query, or built-in framework tools to keep data in sync with the backend.

Monolithic vs. Microservices

- **Monolithic:** Easy to start, less moving parts, but gets hard to manage as the app grows.

- **Microservices:** Breaks the app into smaller pieces. Each piece can be updated and scaled separately, but adds complexity and needs good communication between services.

Modern Trends

- **Component-driven UI:** Build with reusable components for speed and consistency.
- **Edge and Serverless:** Move logic closer to users for speed and reliability.
- **AI-Powered Frontends:** Use AI to personalize UI and automate repetitive tasks[3].

Accessibility Checklist (WCAG 2025)

1. Use Semantic HTML:

Use the right HTML tags for headings, lists, buttons, and forms. This helps screen readers and makes navigation easier[10].

2. Provide Text Alternatives:

All images and non-text content need clear alt text.

3. Ensure Color Contrast:

Text and backgrounds must have enough contrast (at least 4.5:1 for normal text).

4. Keyboard Navigation:

All features must work with just a keyboard (Tab, Enter, etc.). Use skip links to let users jump to main content[11][12][13].

5. Use ARIA Attributes Carefully:

ARIA helps make custom controls accessible, but use it only when you can’t use semantic HTML.

6. Responsive and Mobile-Friendly:

Make sure your site works well on all screen sizes and devices.

7. Test with Real Users and Tools:

Use screen readers, keyboard only, and accessibility checkers (like Axe or Lighthouse) to find issues[10].

Search Query Patterns for Developers

General Tips:

- Always search in English for the best results.
- Use specific error messages or exact code snippets in quotes.
- Add the framework, language, or version to narrow results.
- Use official docs, GitHub, Stack Overflow, and AI-powered search engines (like Perplexity, Phind, or GitHub Copilot)[14][15].

Pattern Examples:

1. **Compare Frameworks:**

   `"Next.js vs SvelteKit 2025 performance comparison"`

   `"React vs Vue pros and cons 2025"`

2. **Find Best Practices:**

   `"best practices authentication Next.js 2025"`

   `"accessibility checklist WCAG 2.1"`

3. **Error Debugging:**

   Paste the full error message in quotes:

   `"TypeError: Cannot read property 'map' of undefined" React 2025`

4. **GitHub Advanced:**

   `todo app language:typescript stars:>100 pushed:>2025-01-01`

   `react language:typescript topic:nextjs stars:>500` [16][17][18][19].

5. **API Reference:**

   `"fetch user data" site:developer.mozilla.org`

   `authentication docs site:nextjs.org`

This cheat sheet gives you the essentials for web development in 2025—embracing the vibe, picking the right tools, staying accessible, and searching smarter.