▯ Vibe Coding Cheat Sheet 2025

Your mental model for building anything from landing pages to full-stack SaaS

▯ Quick Decision Matrix

If you want...	Use this stack	Time to MVP	Why this vibe
Fast MVP	Astro + Tailwind + Netlify	1-3 days	Static site, lightning fast, zero complexity
Learning project	Vue + Vite + Firebase	3-7 days	Gentle learning curve, all-in-one backend
Portfolio/Blog	Next.js + MDX + Vercel	1-2 weeks	SEO-friendly, content-focused, professional
SaaS Product	T3 Stack (Next.js + tRPC + Prisma)	8-12 weeks	Type-safe heaven, scales to millions
E-commerce	Next.js + Stripe + Sanity CMS	4-6 weeks	Payment processing, content management
Real-time app	Next.js + Supabase + Pusher	3-4 weeks	Live updates, PostgreSQL, auth included

▯ Tech Stacks (Plain English)

MERN Stack (Most Popular)

What it is: MongoDB + Express.js + React + Node.js
Plain English: Everything is JavaScript, so you only need to learn one language for your entire app.
Analogy: Like a restaurant where the same chef handles appetizers, main course, and desserts—consistent throughout.
Best for: Startups, rapid prototyping, JavaScript lovers
Vibe: Classic, battle-tested, huge community

T3 Stack (Type-safe God Mode)

What it is: Next.js + TypeScript + tRPC + Prisma + Tailwind
Plain English: Catches mistakes before they become bugs. Everything talks to everything else perfectly.
Analogy: Like having a grammar checker for your code that prevents miscommunication between different parts.
Best for: Complex applications, teams, long-term maintenance
Vibe: Modern, bulletproof, enterprise-ready

JAMstack (Speed Demon)

What it is: JavaScript + APIs + Markup (pre-built HTML)
Plain English: Your site is pre-built and served from a CDN worldwide. Faster than lightning.
Analogy: Like meal prep—cook everything ahead of time, so serving is instant.
Best for: Blogs, marketing sites, documentation
Vibe: Fast, secure, scalable

Frontend Frameworks (5th Grade Explanations)

React

Plain English: React is like building with LEGO blocks. Each block (component) does one job, and you snap them together to build your website.
Real-world analogy: Recipe book where each recipe (component) can be used in multiple meals (pages). Change the recipe once, updates everywhere.
Bundle size: 42KB
Learning curve: Medium
When to use: Large apps, team projects, complex interactions
Magic keywords: "component-based", "virtual DOM", "JSX"

Vue

Plain English: Vue is the friendly framework that's easy to learn. It feels like writing regular HTML but with superpowers.
Real-world analogy: Like a smart assistant that helps you organize your thoughts. It guides you gently without being bossy.
Bundle size: 35KB
Learning curve: Easy
When to use: Quick prototypes, small/medium apps, learning projects
Magic keywords: "progressive", "template-based", "gentle learning curve"

Svelte

Plain English: Svelte does all the heavy work before your site goes live, so it's lightning-fast for users. No extra baggage.
Real-world analogy: Most frameworks pack your entire closet for a trip. Svelte only packs what you actually need.
Bundle size: 1.6KB (smallest!)
Learning curve: Easy
When to use: Performance-critical apps, mobile apps, modern patterns
Magic keywords: "compiler", "no runtime", "blazing fast"

Next.js

Plain English: React with superpowers. Handles the backend, makes your site load faster, and helps Google find you.
Real-world analogy: If React is a car, Next.js is the same car with GPS, automatic transmission, and heated seats already installed.
Bundle size: ~102KB (with features)
Learning curve: Medium-Hard
When to use: Full-stack apps, SEO-critical sites, production apps
Magic keywords: "SSR", "file-based routing", "full-stack"

▯ CSS Frameworks (Style Wars)

Tailwind CSS (Utility-First)

Plain English: Instead of writing custom CSS, you add tiny pre-made classes directly to your HTML. Like paint-by-numbers for websites.
Analogy: Traditional CSS is like writing a novel. Tailwind is like filling in Mad Libs—just pick words from a list.
Bundle size: 3KB (after purging)
Learning curve: Medium
Vibe: Rapid prototyping, design systems, modern

Bootstrap (Component-Based)

Plain English: Pre-built website components (buttons, forms, menus) that look professional out of the box.
Analogy: Like IKEA furniture—pre-designed pieces you can mix and match to furnish your website.
Bundle size: 25KB
Learning curve: Easy
Vibe: Quick MVPs, familiar patterns, corporate look

CSS Modules (Scoped Styles)

Plain English: CSS that only applies to specific components, so styles never accidentally affect other parts.
Analogy: Like having separate wardrobes for different family members—no mix-ups or conflicts.
Bundle size: 0KB (built into Next.js)
Learning curve: Easy
Vibe: Clean architecture, no naming conflicts

🗄 Databases (Plain English)

PostgreSQL (SQL)

Plain English: A digital filing cabinet with strict organization rules. Everything has its proper place and relationships.
Analogy: Like a library with a detailed card catalog system—everything is organized and cross-referenced.
When to use: Complex data relationships, financial data, enterprise apps
Magic keywords: "relational", "ACID compliance", "complex queries"

MongoDB (NoSQL)

Plain English: A flexible storage box where you can throw anything in without strict organization rules.
Analogy: Like a junk drawer—useful for storing random stuff quickly, but can get messy over time.
When to use: Rapid prototyping, flexible schemas, real-time apps
Magic keywords: "document database", "flexible schema", "JSON-like"

SQLite (File-based)

Plain English: A tiny database that lives in a single file. Perfect for small projects or local development.
Analogy: Like a personal notebook versus a massive library system.
When to use: Prototypes, mobile apps, local development
Magic keywords: "serverless", "file-based", "lightweight"

Supabase (Firebase Alternative)

Plain English: PostgreSQL database with built-in authentication, real-time updates, and file storage. Open-source Firebase.
Analogy: Like getting a fully furnished apartment instead of an empty room—everything you need is already there.
When to use: Full-stack apps, real-time features, PostgreSQL lovers
Magic keywords: "Postgres as a service", "real-time", "auth included"

🚀 Deployment Platforms (Getting Live)

Vercel (Next.js Home)

Plain English: Push your code to GitHub, and Vercel automatically publishes your website. Zero setup, just works.
Analogy: Like printing a document. You hit "Print," and the printer does all the work.
Best for: Next.js apps, frontend projects, serverless functions
Magic keywords: "zero-config", "Git integration", "edge functions"

Netlify (JAMstack King)

Plain English: Drag and drop your website folder, and it's live on the internet instantly.
Analogy: Like uploading photos to Instagram—simple drag, drop, share.
Best for: Static sites, serverless functions, forms
Magic keywords: "drag and drop", "form handling", "branch previews"

Railway (Full-stack Friendly)

Plain English: Deploy full-stack apps (frontend + backend + database) with one command.
Analogy: Like a moving company that packs, moves, and unpacks everything for you.
Best for: Full-stack apps, databases, hobby projects
Magic keywords: "full-stack", "database included", "hobby-friendly"

□ Authentication Services (User Login)

Clerk (Developer Experience)

Plain English: Beautiful, customizable login pages that handle all the security stuff automatically.
Analogy: Like hiring a professional bouncer for your club—they handle IDs, guest lists, and troublemakers.
Features: Social login, magic links, webhooks, beautiful UI
Magic keywords: "developer experience", "customizable UI", "webhooks"

Auth0 (Enterprise-Grade)

Plain English: Industrial-strength user management for serious applications.
Analogy: Like a high-security bank vault versus a home safe—overkill for most, perfect for some.
Features: SSO, MFA, extensive integrations, enterprise features
Magic keywords: "enterprise", "SSO", "compliance"

NextAuth.js (Self-hosted)

Plain English: Open-source authentication you host yourself. Full control, no monthly fees.
Analogy: Like cooking at home versus ordering takeout—more work but total control over ingredients.
Features: Many providers, self-hosted, flexible
Magic keywords: "open source", "self-hosted", "many providers"

□ Magic Search Keywords (Get Exact Results)

Speed Keywords

- "**fast landing page**" → Astro + Tailwind + Netlify
- "**blazing fast site**" → Svelte + SvelteKit + Vercel
- "**performance critical**" → Svelte + Preact + edge deployment
- "**mobile first**" → Svelte + Capacitor or React Native

Scale Keywords

- "**enterprise app**" → Next.js + TypeScript + tRPC + PostgreSQL
- "**scalable SaaS**" → T3 Stack + PlanetScale + Vercel
- "**high traffic**" → Next.js + Redis + CDN + load balancing

Learning Keywords

- "**beginner friendly**" → Vue + Vite + Firebase
- "**learning project**" → Svelte + SvelteKit + Supabase
- "**tutorial stack**" → HTML + CSS + vanilla JavaScript

Feature Keywords

- "**real-time chat**" → Next.js + Supabase + WebSockets
- "**user authentication**" → Next.js + Clerk + tRPC
- "**file uploads**" → Next.js + Uploadcare + S3
- "**payment processing**" → Next.js + Stripe + webhooks

▯ Essential Tools (Developer Experience)

Package Managers

- **npm** - Default, slow but reliable
- **pnpm** - Fast, space-efficient (recommended)
- **bun** - Fastest, cutting-edge (experimental)

Code Editors

- **VS Code** - Industry standard, massive extension ecosystem
- **Cursor** - VS Code with AI built-in (2025 trending)
- **WebStorm** - Powerful IDE for complex projects

AI Coding Assistants

- **GitHub Copilot** - Code completion, pair programming
- **Cursor AI** - Context-aware code generation
- **Claude Dev** - Code reviews, architecture advice

🎨 Design Systems & UI Libraries

shadcn/ui (Copy-Paste Components)

Plain English: Beautiful, customizable components you copy into your project. Own your code completely.
Analogy: Like getting recipes instead of pre-made meals—you can modify and improve them.
Best for: Custom design systems, full control, modern React

Chakra UI (React Components)

Plain English: Pre-built React components with great accessibility and theming built-in.
Analogy: Like a toolkit where every tool is designed to work together perfectly.
Best for: React apps, accessibility-first, rapid development

Material-UI (MUI) (Google's Design)

Plain English: Components that follow Google's Material Design rules. Professional, familiar look.
Analogy: Like wearing a business suit—professional and recognizable everywhere.
Best for: Enterprise apps, consistent design, large component library

⚡ Performance Optimization (Making it Fast)

Core Web Vitals (Google's Metrics)

- **LCP** (Largest Contentful Paint) - Main content loads fast (under 2.5s)
- **FID** (First Input Delay) - Clicks/taps respond quickly (under 100ms)
- **CLS** (Cumulative Layout Shift) - Page doesn't jump around (under 0.1)

Optimization Techniques

- **Code splitting** - Load only what users need right now
- **Lazy loading** - Load images/components when user scrolls to them
- **Image optimization** - Use Next.js `<Image>` component, WebP format
- **Bundle analysis** - Find and remove unused code (tree shaking)

🧪 Testing Stack (Making Sure it Works)

Testing Types

- **Unit tests** - Test individual functions (like testing ingredients)
- **Integration tests** - Test components working together (like testing recipes)
- **E2E tests** - Test full user workflows (like testing the complete dining experience)

Testing Tools

- **Vitest** - Fast unit testing (Vite-native)
- **Jest** - Industry standard unit testing
- **Cypress** - End-to-end testing (user simulation)
- **Playwright** - Modern E2E testing (Microsoft)

🔒 Security Essentials (Keeping it Safe)

Common Vulnerabilities

- **XSS** (Cross-site scripting) - Bad actors inject malicious code
- **CSRF** (Cross-site request forgery) - Fake requests from other sites
- **SQL Injection** - Malicious database queries
- **Insecure dependencies** - Outdated packages with security holes

Protection Strategies

- **Input validation** - Check all user input before using it
- **HTTPS everywhere** - Encrypt all connections
- **Regular updates** - Keep dependencies current
- **Environment variables** - Hide API keys and secrets
- **CSP headers** - Control what resources can load

♿ Accessibility Standards (Making it Usable for Everyone)

WCAG 2.1 AA Compliance

- **Color contrast** - 4.5:1 ratio for normal text, 3:1 for large text
- **Keyboard navigation** - Everything clickable must be reachable by Tab key
- **Screen reader support** - Proper ARIA labels and semantic HTML
- **Focus indicators** - Clear visual feedback for keyboard users

Accessibility Quick Checks

- Can you navigate with only the Tab key?
- Are colors still distinguishable in grayscale?
- Do images have descriptive alt text?
- Are form inputs properly labeled?
- Does the site work with screen readers?

Mobile-First Design Principles

Responsive Breakpoints

- **Mobile:** 320px - 768px (design for this first)
- **Tablet:** 768px - 1024px
- **Desktop:** 1024px and up

Touch-Friendly Guidelines

- **Button size:** Minimum 44px x 44px for finger taps
- **Spacing:** At least 8px between clickable elements
- **Text size:** Minimum 16px to prevent zooming
- **Loading states:** Show progress for slow connections

2025 Trends to Watch

AI-First Development

- **GitHub Copilot** - Pair programming with AI
- **v0.dev** - Generate UI from text descriptions
- **Cursor** - IDE with built-in AI assistant
- **Claude Dev** - Code reviews and architecture advice

Edge Computing

- **Vercel Edge Functions** - Run code closer to users
- **Cloudflare Workers** - Global serverless platform
- **Next.js Edge Runtime** - React Server Components at the edge

Type Safety Everywhere

- **TypeScript adoption** - 80%+ of new projects
- **tRPC** - End-to-end type safety for APIs
- **Prisma** - Type-safe database queries
- **Zod** - Runtime type validation

Performance-First Frameworks

- **Astro** - Islands architecture, minimal JavaScript
- **Qwik** - Resumable JavaScript, O(1) loading
- **Fresh** - Deno-based, islands with zero JavaScript by default

Quick Start Commands (Copy-Paste Ready)

```
# Next.js with TypeScript + Tailwind
npx create-next-app@latest my-app --typescript --tailwind --app

# Vite + React + TypeScript
npm create vite@latest my-app -- --template react-ts

# Astro (fastest static sites)
npm create astro@latest

# SvelteKit
npm create svelte@latest my-app

# T3 Stack (full-stack TypeScript)
npm create t3-app@latest

# Vue 3 + Vite
npm create vue@latest my-project
```

Deployment Commands (Going Live)

```
# Vercel (recommended for Next.js)
npx vercel

# Netlify
npm install -g netlify-cli
netlify deploy

# Build and deploy to any static host
npm run build
# Upload dist/ or build/ folder

# Docker (for full-stack apps)
docker build -t my-app .
docker run -p 3000:3000 my-app
```

Pro Tips for 2025

Development Workflow

1. **Start small** - Begin with Create React App or Vite, add complexity later
2. **TypeScript from day one** - Saves debugging time later
3. **Mobile-first CSS** - Design for phones, then scale up
4. **Git commits early and often** - Small, descriptive commits
5. **Test user flows** - Focus on what users actually do

Performance Mindset

1. **Measure first** - Use Lighthouse, Core Web Vitals
2. **Images are usually the bottleneck** - Optimize aggressively
3. **Bundle size matters** - Especially on mobile connections
4. **Caching is king** - Both browser and CDN caching
5. **Progressive enhancement** - Basic functionality first, fancy features second

Deployment Strategy

1. **Preview deployments** - Test before going live (Vercel/Netlify)
2. **Environment variables** - Different configs for dev/staging/prod
3. **Monitoring** - Know when things break (Sentry, LogRocket)
4. **Rollback plan** - Always have a way back to the previous version
5. **Performance budgets** - Set limits on bundle size, load time

When in Doubt, Use This

Building your first project? → Vue + Vite + Firebase
Need it fast? → Astro + Tailwind + Netlify
Long-term project? → Next.js + TypeScript + Supabase
Learning? → Follow a tutorial with vanilla HTML/CSS/JS first

Remember: **The best stack is the one you can ship with.** Start simple, add complexity only when needed. Perfect is the enemy of good—launch something, then make it better!

This cheat sheet is your mental model for vibe coding in 2025. Mix and match these stacks, keywords, and patterns to build anything from landing pages to full-stack SaaS products. Keep it simple, keep it accessible, keep shipping!