Vibe Coding

What is vibe coding?

The vibe coder mindset

AI Assisted Coding Tools

- Claude Code
- Gemini
- ChatGPT
- Cursor
- Windsurf
- Replit

Frontend-Focused

- v0
- Lovable

Find the detailed version of this roadmap along with other similar roadmaps

roadmap.sh

Related Roadmaps

- Claude Code Roadmap
- AI Engineer Roadmap
- AI Agents Roadmap

Plan before you Code

- [ ] Plan what you need to develop (MVP, Different Phases)
- [ ] Work step by step rather than trying to build everything at once
- [ ] Illustrate AI with examples (mockups, code samples, images)

AI Tools can Help with Planning

For example if you are using "Claude Code", tell the tool what you are trying to build, ask it to help refine the idea, establish the different phases, and once done ask it to document everything in a document so that you can refer to that when actually building the product.

Tech Stack and Coding

- [ ] Pick a popular tech stack rather than new/niche ones
- [ ] If you have style/coding preferences, document them for AI
- [ ] Ask AI to keep the code modular and aim for smaller modules/files
- [ ] Regularly ask the AI to review and refactor the codebase
- [ ] Use skills created by others

Establishing Standards Early is Important

When you're starting out, carefully review the AI's initial outputs - coding patterns, styles, architecture decisions. If you don't catch bad habits early, the AI will reinforce them with every iteration, and they'll compound fast.

Force refactoring sessions regularly

AI will take the path of least resistance (appending code, growing files, skipping cleanup). Periodically ask it to step back and refactor: break things into smaller modules, remove dead code, improve performance.

Prompting Best Practices

- [ ] Ask for one task at a time rather than five different items.
- [ ] Be specific about what you want, rather than high-level, vague instructions
- [ ] Based on your previous coding sessions, tell AI what NOT to do
- [ ] Give AI mockups, reference files and material that can help it
- [ ] Use "act as" framing when helpful (e.g. act as a UX researcher)
- [ ] Regularly update your context document (e.g. CLAUDE.md)
- [ ] Explicitly tell AI to "think" or "brainstorm" before complex problems

Keep Context Document Up to Date

If you see yourself repeating some instructions, document the instructions in the context document. One more thing that helps is: after a discussion session if you feel like AI can benefit from learnings in the session, ask it to update the context document (e.g. claude.md) with the learnings from the session.

Context

- [ ] Leverage long context window when available and necessary
- [ ] If AI fails after 3 prompts, stop, and start a fresh chat
- [ ] For unrelated tasks, proactively clean and start new sessions
- [ ] Ask AI to use subagents, if possible

Clear Context Regularly

Clear context for better results and to save token costs. Whenver you are about to start a new/unrelated task, it's better to clear the context and start fresh.

Debugging

- [ ] Prompt the error message and let AI do the rest
- [ ] If errors persist, Ask AI to create a list of possible causes
- [ ] Tell AI to add logs to find the error faster
- [ ] Install and ask AI to use MCP (e.g. Playwright for browser), when possible

Let AI Debug, But Understand what Went Wrong

When something breaks, paste the error and relevant code and ask AI to explain the error. Don't just accept the fix — make sure you understand it. If you don't, ask AI it to explain in simple terms.

Master Version Control

- [ ] Use `git commit` regularly (e.g. after every successful AI task)
- [ ] Start each new feature with a clean Git slate
- [ ] If you need to revert, use Git rather than AI native revert functionality
- [ ] Ask AI to handle your Git and GitHub CLI tasks

Commit Often, With Clear Messages

After each working feature or fix, make a commit. Ask AI to suggest a clear commit message that describes what changed and why. This gives you a safe checkpoint to roll back to if something goes wrong later.

Testing

- [ ] Ask AI to write tests (E2E tests can help build a stable product)
- [ ] Consider Test-driven development (TDD)
- [ ] When you find a bug, ask AI to write a breaking test and then fix
- [ ] Once tests are in place, refactor regularly

Force AI To Test by Default

Most AI tools' default behavior produces implementation-first code with minimal test coverage. Whenever AI builds a feature, force it to write basic tests right away. This way bugs get caught early and you always know when something new breaks something old.

Security Best Practices

- [ ] Explicitly ask AI to perform a security audit of the application
- [ ] Never hardcode or credentials; use env variables instead

Never Hardcode your Secrets

Never let the AI put passwords, API keys, or tokens directly in the code. If you see it doing this, stop it and ask it to use environment variables instead. This is one of the most common beginner mistakes and one of the most dangerous.

Continue Learning with following relevant tracks

- Claude Code
- AI Engineer
- AI Agents