AI-Powered Web Development: Vibe-Coding with Windsurf, GitHub, and Vercel

Table of Contents

What is Vibe-Coding?

Vibe-coding is a modern development approach where you use natural language prompts to guide AI in building software, rather than writing every line of code manually. With Windsurf's Cascade AI, you describe what you want, and the AI generates the implementation.

Traditional vs Vibe-Coding

Traditional	Vibe-Coding with AI
Write code manually	Describe what you want
Search Stack Overflow	Ask AI for solutions
Debug for hours	AI suggests fixes instantly
Read documentation	AI explains and implements
Slow iteration	Rapid prototyping

Why It Works

- **Speed**: Build prototypes in minutes, not hours
- **Learning**: AI explains concepts as it codes
- **Exploration**: Try ideas without deep technical knowledge
- **Focus**: Concentrate on design and user experience

Setting Up Your Project Folder in Windsurf

Creating a New Folder in the Documents Directory

Before starting your project, you need to create a dedicated folder to store all your files. Here's how to do it within Windsurf:

Method 1: Using the Explorer Panel (Recommended)

1. **Open Windsurf**
   - Launch the Windsurf IDE from your desktop or Start menu
2. **Access the Explorer Panel**
   - Look for the file icon in the left sidebar (or press `Ctrl/Cmd + Shift + E`)
   - This shows your current workspace structure
3. **Navigate to Documents**
   - Click on `File` → `Open Folder`
   - Navigate to `C:\Users\YourUsername\Documents` (Windows) or `/Users/YourUsername/Documents` (Mac)
   - Click `Select Folder` or `Open`
4. **Create Your Project Folder**
   - Right-click in the Explorer panel on the Documents folder
   - Select `New Folder` from the context menu
   - Type your project name (e.g., `my-website`)
   - Press `Enter` to confirm
5. **Open the New Folder**
   - Click `File` → `Open Folder` again
   - Navigate to your newly created folder
   - Click `Open`
   - Windsurf will reload with your project folder as the workspace

Method 2: Using the Terminal

1. **Open Terminal in Windsurf**
   - Press `Ctrl+`` (backtick) or go to `Terminal` → `New Terminal`
   - The terminal appears at the bottom of the screen
2. **Create the Folder**

```
# Navigate to Documents
cd ~/Documents

# Create project folder
mkdir my-website

# Navigate into the folder
cd my-website

# Open this folder in Windsurf
code .
```

3. **Windsurf Opens the Folder**
    - The folder is now your active workspace
    - You're ready to start your project

Method 3: Using Cascade (AI)

1. **Open Cascade Panel**
    - Press `Ctrl/Cmd + L`
2. **Ask AI to Create the Folder**

```
Create a new folder called "my-website" in my Documents directory
and open it as the current workspace.
```

3. **Cascade Will:**
    - Create the folder for you
    - Set it as the active workspace
    - Confirm the action

Managing Projects with Workspace Files

What is a Workspace File?

A workspace file (`.code-workspace`) is a configuration file that saves your project settings, open files, and folder structure. It allows you to quickly reopen entire projects with all your preferences intact.

Benefits of Using Workspace Files

- **Quick Access**: Open previous projects instantly
- **Saved State**: Remember open files, terminal history, and settings
- **Multiple Projects**: Switch between different projects easily
- **Consistency**: Same environment every time you open the project
- **Team Sharing**: Share workspace configurations with teammates

Creating a Workspace File

Method 1: Save Current Workspace

1. **Open Your Project Folder**
    - Ensure your project is loaded in Windsurf
    - Check the Explorer panel shows your project files
2. **Save Workspace**
    - Click `File` → `Save Workspace As...`
    - Navigate to a location (e.g., `Documents/Workspaces/`)
    - Name the file: `my-project.code-workspace`
    - Click `Save`
3. **Workspace File Created**
    - A `.code-workspace` file is saved

- Contains paths to your project folders and settings

Method 2: Create from Scratch

1. **Create New Workspace**
   - Click `File` → `Open Workspace from File...`
   - Or `File` → `New Window` → then add folders
2. **Add Folders to Workspace**
   - Click `File` → `Add Folder to Workspace`
   - Select your project folder
   - Repeat for multiple projects
3. **Save the Workspace**
   - `File` → `Save Workspace As...`
   - Name and save the file

Opening a Previous Project

Option 1: Double-Click Workspace File

1. Navigate to where you saved `.code-workspace` file
2. Double-click the file
3. Windsurf opens with your complete project setup

Option 2: From Within Windsurf

1. Click `File` → `Open Workspace from File...`
2. Navigate to your `.code-workspace` file
3. Click `Open`
4. Your project loads with all previous settings

Option 3: Recent Workspaces

1. Click `File` → `Open Recent`
2. Select from list of recent workspaces
3. Project opens instantly

Workspace File Structure

A workspace file looks like this:

```
{
  "folders": [
    {
      "path": "my-website"
    }
  ],
  "settings": {
    "editor.fontSize": 14,
    "editor.tabSize": 2
  },
  "extensions": {
    "recommendations": [
      "esbenp.prettier-vscode",
```

```
      "bradlc.vscode-tailwindcss"
    ]
  }
}
```

Best Practices for Workspaces

1. **Organize by Client/Project**

```
Workspaces/
├── client-a-website.code-workspace
├── personal-blog.code-workspace
├── ecommerce-project.code-workspace
└── tutorial-learning.code-workspace
```

2. **Include Multiple Related Folders**
   - Frontend and backend in same workspace
   - Shared components library + main project
3. **Version Control Workspaces**
   - Commit `.code-workspace` files to Git
   - Team members get same setup
4. **Customize Per Project**
   - Different color themes for different clients
   - Specific extensions for project type

Project Planning Before Opening Windsurf

Why Plan Before Coding?

Planning before opening your IDE helps you:

- Clarify project goals and requirements
- Make better use of AI assistance
- Avoid rework and refactoring
- Stay focused on the end result
- Communicate clearly with Cascade

The Pre-Development Checklist

Step 1: Define the Project Purpose

Questions to Answer:

- What problem does this project solve?
- Who is the target audience?
- What is the primary goal? (inform, sell, entertain, educate)
- What makes this project unique?

Document Your Answers:

```
Project: Portfolio Website
Purpose: Showcase my work to potential clients
Audience: Small business owners looking for web design
Goal: Generate leads through contact form
Unique Factor: Interactive project gallery with live demos
```

Step 2: Plan the Content Structure

Outline Your Pages:

```
Home
├── Hero section (headline, CTA)
├── Services overview
├── Featured projects (3-4 items)
├── Testimonials
└── Contact CTA

About
├── Bio/Story
├── Skills/Tech stack
├── Experience timeline
└── Download resume button

Portfolio
├── Project grid
├── Filter categories
├── Project detail modals
└── Live demo links

Contact
├── Contact form
├── Social links
├── Email/phone info
└── Location map
```

List Required Assets:

- Logo (SVG or PNG)
- Project screenshots (16:9 ratio)
- Profile photo (square)
- Resume PDF
- Favicon

Step 3: Define the Design System

Color Palette:

```
Primary: #DC2626 (Red)
Secondary: #1F2937 (Dark Gray)
Accent: #10B981 (Green for success)
Background: #FFFFFF (White)
Text: #111827 (Near Black)
```

Typography:

```
Headings: Inter, sans-serif
Body: Inter, sans-serif
Code: JetBrains Mono, monospace
```

Component Ideas:

- Navigation: Sticky header, mobile hamburger
- Buttons: Primary (filled), Secondary (outline)
- Cards: Rounded corners, subtle shadow
- Forms: Clean labels, inline validation

Step 4: Technical Decisions

Framework & Tools:

- Next.js with App Router
- TypeScript for type safety
- Tailwind CSS for styling
- shadcn/ui for components
- Framer Motion for animations

Integrations:

- Contact form → Email service (Resend/Formspree)
- Analytics → Google Analytics or Plausible
- Hosting → Vercel
- Domain → Custom domain (optional)

Data Structure:

```typescript
// Project data structure
interface Project {
  id: string;
  title: string;
  description: string;
  image: string;
  tags: string[];
  liveUrl: string;
  githubUrl?: string;
}
```

Step 5: Create a Prompt Strategy

Prepare Initial Prompt:

```
Create a portfolio website for a web developer with:
- Dark red (#DC2626) primary color
- Clean, modern design
- Hero section with "John Doe - Full Stack Developer"
- Services: Web Design, Development, SEO
- Portfolio grid with 6 projects
- Contact form with validation
```

```
- Responsive design
- Dark mode support
```

Plan Follow-Up Prompts:

1. "Add animations to the hero section"
2. "Create project detail modals"
3. "Add form validation and error handling"
4. "Optimize for SEO with meta tags"
5. "Add loading states and transitions"

Step 6: Set Up Your Environment

Before Opening Windsurf:

1. ✅ Project folder created
2. ✅ Planning documents ready
3. ✅ Assets collected in project folder
4. ✅ Color palette documented
5. ✅ Content outline complete

Organize Assets:

```
my-website/
├── assets/
│   ├── logo.svg
│   ├── profile-photo.jpg
│   ├── project-1.jpg
│   ├── project-2.jpg
│   └── resume.pdf
└── planning/
    ├── content-outline.md
    ├── design-system.md
    └── prompt-strategy.md
```

Using Your Plan with Cascade

Start with Context:

```
I'm building a portfolio website. Here's my plan:
- Target: Small business clients
- Pages: Home, About, Portfolio, Contact
- Style: Modern, professional, red accent color
- Features: Dark mode, contact form, project gallery

Let's start by creating the project structure and home page.
```

Reference Your Plan:

```
According to my design system (primary color #DC2626),
create a navigation component with:
- Logo on left
- Links: Home, About, Portfolio, Contact
```

```
- CTA button "Hire Me" in red
- Mobile hamburger menu
```

Quick Reference: Planning Template

```
# Project Plan: [Name]

## Purpose
- Problem: [What it solves]
- Audience: [Who it's for]
- Goal: [What success looks like]

## Pages & Content
1. [Page Name]
   - [Section 1]
   - [Section 2]

## Design
- Primary: #[COLOR]
- Style: [Modern/Classic/Playful]
- Fonts: [Font names]

## Tech Stack
- Framework: Next.js
- Styling: Tailwind CSS
- Components: shadcn/ui
- Animations: Framer Motion

## Assets Needed
- [ ] Logo
- [ ] Images
- [ ] Copy/text

## Prompt Strategy
1. [First prompt]
2. [Second prompt]
3. [Third prompt]
```

Understanding the Technologies

Before diving into development, let's understand the key technologies that power your workflow:

Next.js

What is Next.js? Next.js is a React framework that enables server-side rendering, static site generation, and full-stack web applications. It provides the foundation for building modern, fast, and SEO-friendly websites.

Key Features:

- **App Router**: Modern routing system with nested layouts
- **Server Components**: Render components on the server for better performance

- **Static Generation**: Pre-build pages for instant loading
- **API Routes**: Build backend endpoints within your Next.js app
- **Image Optimization**: Automatic image resizing and optimization
- **TypeScript Support**: Built-in type safety

Why Use It?

- Faster page loads through server-side rendering
- Better SEO than traditional single-page apps
- Automatic code splitting for smaller bundles
- Developer-friendly with hot reloading
- Production-ready out of the box

Example:

```
// app/page.tsx
export default function Home() {
  return (
    <main>
      <h1>Welcome to Next.js</h1>
    </main>
  )
}
```

Git

What is Git? Git is a distributed version control system that tracks changes in your code over time. It allows you to save snapshots of your project, collaborate with others, and revert changes when needed.

Key Concepts:

- **Repository**: A folder containing your project and its history
- **Commit**: A snapshot of your files at a specific point in time
- **Branch**: An independent line of development
- **Staging**: Preparing files to be committed
- **History**: Complete record of all changes made

Why Use It?

- Track every change to your code
- Experiment safely with new features
- Collaborate without overwriting others' work
- Revert mistakes instantly
- Maintain multiple versions of your project

Essential Commands:

```
# Start tracking a project
git init

# Save changes with a message
git add .
git commit -m "Added hero section"
```

```
# View history
git log

# Check current status
git status
```

GitHub

What is GitHub? GitHub is a web-based platform that hosts Git repositories. It provides a centralized place to store your code, collaborate with others, and manage projects.

Key Features:

- **Remote Repositories**: Store code in the cloud
- **Collaboration Tools**: Pull requests, code reviews, issues
- **Actions**: Automated workflows for testing and deployment
- **Pages**: Host static websites directly from repositories
- **Teams**: Organize contributors and permissions

Why Use It?

- Backup your code safely in the cloud
- Share projects with others
- Collaborate on team projects
- Showcase your work to potential employers
- Integrate with deployment platforms like Vercel

Workflow:

```
# Connect local to remote
git remote add origin https://github.com/username/repo.git

# Upload changes
git push -u origin main

# Download changes
git pull origin main
```

Vercel

What is Vercel? Vercel is a cloud platform for static sites and serverless functions. It provides instant deployment, automatic scaling, and global CDN distribution.

Key Features:

- **Zero-Config Deployment**: Push to GitHub, auto-deploy
- **Preview URLs**: Every branch gets its own URL
- **Serverless Functions**: API endpoints without managing servers
- **Edge Network**: Global CDN for fast loading worldwide
- **Analytics**: Built-in performance monitoring
- **Environment Variables**: Secure configuration management

Why Use It?

- Deploy websites with a single `git push`
- Automatic HTTPS and custom domains
- Instant rollbacks to previous versions
- Preview deployments for testing
- Optimized for Next.js applications
- Free tier for personal projects

Deployment Process:

1. Connect GitHub repository to Vercel
2. Push code to main branch
3. Vercel automatically builds and deploys
4. Site is live at `your-project.vercel.app`

Windsurf IDE

What is Windsurf? Windsurf is an AI-powered integrated development environment (IDE) that combines code editing with artificial intelligence assistance. It's built on top of VS Code with added AI capabilities through Cascade.

Key Features:

- **Cascade AI**: Natural language coding assistant
- **Codeium Integration**: AI-powered autocomplete
- **VS Code Compatibility**: All VS Code extensions work
- **Integrated Terminal**: Command line access within the editor
- **File Explorer**: Visual file and folder management
- **Debugger**: Built-in debugging tools
- **Git Integration**: Visual diff and commit tools

Why Use It?

- Write code faster with AI assistance
- Get instant help with errors and bugs
- Generate code from natural language descriptions
- Refactor and improve code with AI suggestions
- Explain complex code in plain English
- All the power of VS Code plus AI

Cascade in Action:

```
User: "Create a navigation component with mobile hamburger menu"

AI: Generates complete React component with:
- Desktop navigation bar
- Mobile hamburger menu
- Responsive design
- TypeScript types
- Styling with Tailwind CSS
```

Essential Shortcuts:

- `Ctrl/Cmd + L` - Open Cascade AI panel
- `Ctrl/Cmd + Shift + E` - Open Explorer

- `Ctrl+`` - Open Terminal
- `Ctrl/Cmd + P` - Quick file navigation
- `Ctrl/Cmd + Shift + F` - Search across files

The AI Prompting Workflow

Understanding Cascade (Windsurf's AI)

Cascade is your AI pair programmer. Access it:

- **Shortcut**: `Ctrl/Cmd + L`
- **Command**: Type `@` to reference files
- **Context**: AI sees your current file and workspace

The Vibe-Coding Loop

```
1. PROMPT → Describe what you want
2. REVIEW → AI suggests changes
3. ACCEPT → Apply the changes
4. TEST → See if it works
5. ITERATE → Refine with follow-up prompts
```

Optimizing Your Windsurf Workspace for Learning

Setting Up a Side-by-Side View: Code + Live Preview

One of the most powerful ways to learn vibe-coding is to see changes happen in real-time. By setting up a split-screen view, you can watch Cascade make changes while simultaneously seeing those changes reflected in your local website.

Method 1: Built-in Browser Preview (Recommended)

Windsurf has a built-in feature to show your website right alongside your code:

Setup Steps:

1. Ensure your dev server is running (`npm run dev` in terminal)
2. Click the **Preview** icon in the left sidebar (looks like a browser window)
3. Enter your local URL: `http://localhost:3000`
4. The browser preview opens in a panel next to your code
5. Arrange panels: Drag the preview tab to the right side of your screen
6. Now you have code on the left, live website on the right

Alternative Method:

1. Press `Ctrl/Cmd + Shift + P` to open Command Palette
2. Type "Simple Browser: Show"
3. Enter `http://localhost:3000`
4. The browser opens inline

Method 2: External Browser + Windsurf Split

For larger screens or dual-monitor setups:

Single Monitor Setup:

1. Open Windsurf in full screen
2. Open your external browser (Chrome, Edge, Firefox)
3. Arrange windows side-by-side:
    - Windows: Press `Win + Left Arrow` on Windsurf, `Win + Right Arrow` on browser
    - Mac: Use Rectangle app or native split view
4. Navigate browser to `http://localhost:3000`
5. Keep Cascade panel open in Windsurf

Dual Monitor Setup (Ideal for Learning):

- **Monitor 1**: Windsurf with Cascade panel open
- **Monitor 2**: Browser showing `http://localhost:3000`
- This gives maximum space for both coding and viewing

Method 3: Cascade Panel + Browser Split

The most immersive learning setup:

1. Open Cascade panel: `Ctrl/Cmd + L`
2. Resize Cascade panel to take up 40% of screen width
3. Open built-in browser preview on the right 60%
4. Your layout: [Cascade Chat | Browser Preview]

Why This Setup Accelerates Learning:

- **Immediate Feedback**: Type a prompt → See AI work → Watch website update
- **Pattern Recognition**: You start recognizing what certain prompts produce
- **Error Understanding**: When something breaks, you see it immediately
- **Design Intuition**: Visual changes teach design principles faster than reading

---

Learning by Reading AI Responses: Understanding the "Why"

Don't just accept AI changes—read Cascade's explanations to understand **how** vibe-coding works. This transforms you from a passive user into an active learner.

What to Look For in AI Responses

1. File Structure Explanations When Cascade creates files, it often explains:

```
"I created `app/components/navbar.tsx` as a separate component so it can be
reused across pages and tested independently."
```

Learn: Why components are modular

2. Code Pattern Explanations

```
"I used the `useState` hook here to track the mobile menu open/close state.
This is a React pattern for managing UI state."
```

Learn: React concepts and patterns

3. Library Choices

```
"I installed `framer-motion` because it provides declarative animations
that are easier to customize than CSS animations."
```

Learn: Why certain tools are chosen

4. Styling Decisions

```
"I used `flex-col md:flex-row` to make the navigation stack vertically on
mobile and horizontally on desktop. This is mobile-first responsive design."
```

Learn: Tailwind CSS responsive patterns

5. Error Explanations

```
"The build failed because `Button` wasn't imported. Next.js requires all
components to be explicitly imported. I added the missing import."
```

Learn: Common errors and fixes

Active Reading Exercise

Try this during your next session:

1. **Before Prompting**: Write down what you expect Cascade to do
2. **After Response**: Read Cascade's explanation carefully
3. **Compare**: Did Cascade do what you expected? What did you miss?
4. **Question**: Ask follow-up "Why did you..." prompts

Example Learning Session:

```
You: "Create a dark mode toggle"

Cascade: [Creates toggle component]

Cascade explains: "I used `next-themes` because it handles:
- System preference detection
- Manual override
- No flash on load
- SSR compatibility"

You: "Why is SSR compatibility important?"

Cascade: "Server-Side Rendering means the page loads with the correct
theme already applied, preventing a flash of wrong-colored content."
```

Result: You learned about SSR, theme flashing, and the `next-themes` library

Building Your Mental Model

As you read AI responses, you're building intuition for:

Concept	What You'll Learn
File Organization	Where different types of code belong

Component Design	When to split components vs. keep together
State Management	How React tracks changing data
Styling Patterns	Responsive design, color systems, spacing
Error Handling	Common mistakes and solutions
Performance	What makes code fast or slow
Best Practices	Industry standards and conventions

The "Explain Like I'm 5" Technique

When you don't understand something in Cascade's response:

```
"Can you explain what `useEffect` does in this component?
I'm new to React hooks."
```

Cascade will simplify the concept, often with analogies:

```
"Think of `useEffect` like a robot that watches for changes. When the
`theme` variable changes, the robot springs into action and updates
the document body class."
```

Document Your Learnings

Create a personal "Vibe-Coding Glossary" file:

```
# My Vibe-Coding Learnings

## Terms I Learned
- **SSR**: Server-Side Rendering - page loads complete from server
- **Hydration**: React takes over the server-rendered HTML
- **Tailwind**: Utility-first CSS framework
- **Hook**: React function that adds special features to components

## Patterns I Recognize
- Mobile-first: Write mobile styles, add `md:` for desktop
- Component props: Use `interface` to define what data a component accepts
- API routes: Files in `app/api/` become backend endpoints

## Common Fixes
- "Cannot find module" → Missing import statement
- "Property does not exist" → TypeScript type issue
- Hydration mismatch → Server and client render different HTML
```

The Ideal Learning Workflow

Combine side-by-side viewing with active reading:

```
┌─────────────────────────────────────────────────────────┐
│  CASCADE PANEL    │    BROWSER PREVIEW    │    CODE EDITOR   │
```

```
┌──────────────────┬──────────────────────┬──────────────────┐
│                  │                      │                  │
│  Your prompt:    │   http://localhost   │   page.tsx       │
│  "Add a hero..." │   :3000              │                  │
│                  │                      │                  │
│  ────────────────┼──────────────────────┼──────────────    │
│                  │                      │                  │
│  AI Response:    │   [Live website      │   [Code being    │
│  "I created...   │    showing           │    edited]       │
│   Here's why..." │    changes]          │                  │
│                  │                      │                  │
└──────────────────┴──────────────────────┴──────────────────┘
```

The Cycle:

1. Read the AI's explanation (Cascade panel)
2. See the result (Browser preview)
3. Study the code changes (Code editor)
4. Ask follow-up questions if confused
5. Iterate with new prompts

Why This Works:

- **Visual + Verbal**: You see AND read about changes
- **Immediate**: No delay between action and understanding
- **Iterative**: Each prompt builds on previous knowledge
- **Safe**: Mistakes are local, not live

Pro Tips for Maximum Learning

1. Start with Small Prompts Instead of: "Build me a website" Try: "Add a red button with rounded corners"

Smaller changes = easier to understand what the AI did

2. Compare Before and After Use Windsurf's source control panel to see exactly what changed:

- Click Source Control icon (branch symbol)
- See line-by-line changes
- Click each file to view the diff

3. Save "Aha Moments" When Cascade explains something that clicks:

- Screenshot the explanation
- Add to your learning glossary
- Reference it in future projects

4. Teach the AI When you understand something, reinforce it:

```
"So if I understand correctly, `useState` creates a variable
that React watches for changes, and when it changes, React
re-renders the component. Is that right?"
```

5. Break Things on Purpose Ask Cascade to make changes, then ask why they work:

```
"Can you change the button color to blue and explain
what CSS property you modified?"
```

6. Connect Concepts When you see a pattern multiple times, ask:

```
"I noticed you always put reusable components in a
`components/` folder. Is that a standard convention?"
```

Troubleshooting the Learning Process

"I don't understand what Cascade just did" → Ask: "Can you explain that in simpler terms?"

"Changes are happening too fast to see" → Ask: "Can you make changes one at a time so I can follow?"

"I'm overwhelmed by the code generated" → Start with a fresh project and make one small change at a time

"The AI response is too technical" → Ask: "Explain like I'm new to web development"

Project Lifecycle with AI

Phase 1: Project Initialization with AI

Prompt:

```
Create a new Next.js project with:
- TypeScript support
- Tailwind CSS for styling
- App Router enabled
- shadcn/ui components
- Dark mode support
- A hero section on the home page
- Navigation with links to Home, About, and Contact
```

What Cascade Does:

1. Runs `npx create-next-app@latest`
2. Configures TypeScript and Tailwind
3. Installs shadcn/ui
4. Sets up dark mode with next-themes
5. Creates initial page components
6. Builds a responsive navigation

Your Action:

- Review each file Cascade creates
- Accept or reject changes
- Ask for modifications if needed

Phase 2: Building Features with Prompts

Example 1: Creating a Hero Section

Prompt:

```
Create a hero section for the home page with:
- Large headline: "Build Faster with AI"
- Subheadline explaining our service
```

```
- Call-to-action button "Get Started"
- Background gradient animation
- Responsive design for mobile
- Use framer-motion for subtle entrance animations
```

Cascade Generates:

- `app/components/hero.tsx`
- Animations with Framer Motion
- Responsive Tailwind classes
- Gradient background component

Example 2: Building a Contact Form

Prompt:

```
Create a contact form on the contact page with:
- Fields: name, email, message
- Form validation using React Hook Form and Zod
- Submit button with loading state
- Success message after submission
- Send data to /api/contact endpoint
- Styled with Tailwind and shadcn Input components
```

Cascade Generates:

- Form component with validation
- API route for form submission
- Error handling
- Loading states
- Success feedback

Example 3: Adding a Portfolio Gallery

Prompt:

```
Add a portfolio section to showcase projects:
- Grid layout with 3 columns on desktop, 1 on mobile
- Each card shows project image, title, description
- Hover effects with scale and shadow
- Click to open project details in a modal
- Use project data from a projects.ts file
- Include featured project: Raccoon Muncher game at raccoon-muncher.vercel.app
```

Cascade Generates:

- Portfolio grid component
- Project card with hover effects
- Modal for project details
- Data structure for projects
- Responsive grid layout

Phase 3: Styling and Refinement

Prompt for Design System

Prompt:

```
Create a consistent design system:
- Color palette: primary red (#DC2626), dark gray for backgrounds
- Typography scale using Tailwind defaults
- Button variants: primary (red), secondary (outline), ghost
- Card component with border and hover effects
- Spacing consistency (4px grid)
- Apply to all existing components
```

Cascade:

- Updates `tailwind.config.ts` with custom colors
- Creates reusable button component
- Standardizes spacing across components
- Applies design system to existing pages

Prompt for Responsive Design

Prompt:

```
Make the entire website fully responsive:
- Mobile-first approach
- Navigation becomes hamburger menu on mobile
- Hero text scales down on smaller screens
- Grid layouts collapse to single column
- Test breakpoints: sm (640px), md (768px), lg (1024px)
- Ensure no horizontal scrolling
```

Phase 4: Adding Interactivity

Prompt for Animations

Prompt:

```
Add scroll animations throughout the site:
- Fade in sections as user scrolls
- Stagger animation for list items
- Smooth scroll to sections
- Use Framer Motion or AOS library
- Keep animations subtle and professional
```

Prompt for State Management

Prompt:

```
Add a theme toggle (light/dark mode):
- Toggle button in navigation
- Persist user preference in localStorage
- Apply dark: variants to all components
- Smooth transition between themes
- Use next-themes for SSR compatibility
```

Advanced AI Prompting Techniques

1. Progressive Disclosure

Start broad, then narrow down:

Step 1 - Broad:

```
Create an about page for my digital agency
```

Step 2 - Specific:

```
Add to the about page:
- Team section with 3 member cards
- Each card has photo, name, role, and short bio
- Photos use placeholder avatar
- Grid layout: 3 columns desktop, 1 column mobile
```

Step 3 - Polish:

```
Add hover effects to team cards:
- Scale up slightly on hover
- Show social icons (LinkedIn, Twitter) on hover
- Smooth transition animations
```

2. Reference Existing Code

Use `@` to point to specific files:

```
Update @app/page.tsx to include a testimonials section below the hero.
Use the same card style as @components/ui/card.tsx.
```

3. Context-Aware Prompts

Cascade remembers previous context:

```
Follow-up: Also add a pricing section with 3 tiers:
- Basic: $99/month
- Pro: $199/month
- Enterprise: Contact us

Use the same styling as the features section we just created.
```

4. Debug with AI

When something breaks:

Prompt:

```
The build is failing with this error:
"Type error: Cannot find name 'Badge'"
```

```
Fix the import issue in @client/src/pages/home.tsx
```

Cascade:

- Identifies the missing import
- Adds `import { Badge } from "@/components/ui/badge"`
- Explains why the error occurred

5. Refactoring Prompts

Prompt:

```
Refactor the navigation component:
- Extract mobile menu into separate component
- Use custom hook for menu state
- Add TypeScript interfaces
- Keep all existing functionality
```

From Prompt to Production

The Complete AI-Guided Workflow

Step 1: Initial Setup (AI-Powered)

```
Prompt: "Initialize a Next.js project with all the modern defaults"
```

Cascade Actions:

- Creates project structure
- Sets up TypeScript
- Configures Tailwind
- Initializes shadcn/ui
- Creates basic layout

Step 2: Build Core Pages (Iterative Prompting)

```
Prompt 1: "Create a landing page with hero, features grid, and CTA"
Prompt 2: "Add an about page with company story and team"
Prompt 3: "Build a contact page with form and map"
```

Step 3: Add Features (Specific Prompts)

```
Prompt: "Add a blog section with:
- List of posts on /blog
- Individual post pages at /blog/[slug]
- MDX support for rich content
- Syntax highlighting for code blocks"
```

Step 4: Polish and Optimize

```
Prompt: "Optimize the site for production:
- Add SEO meta tags to all pages
- Generate sitemap.xml
- Add loading states
- Optimize images with next/image
- Add analytics integration"
```

Step 5: Deploy to Vercel

```
Prompt: "Help me deploy this to Vercel:
- Connect to GitHub repository
- Set up automatic deployments
- Configure environment variables
- Add custom domain (optional)"
```

Cascade guides you through:

1. Creating GitHub repo
2. Pushing code
3. Importing to Vercel
4. Configuring settings
5. Deploying

Best Practices for AI Collaboration

DO:

✅ **Be Specific**

```
Good: "Create a blue primary button with white text, rounded corners,
       that changes to darker blue on hover"

Vague: "Make a button"
```

✅ **Provide Context**

```
Good: "Following the design system in @tailwind.config.ts,
       create a card component with our brand colors"
```

✅ **Iterate Incrementally**

```
Prompt 1: "Create basic form structure"
Prompt 2: "Add validation to the form"
Prompt 3: "Style the form errors"
```

✅ **Review AI Output**

- Check for security issues
- Verify TypeScript types
- Test functionality

✅ **Learn from AI**

```
Prompt: "Explain how the useEffect hook works in the component you just created"
```

DON'T:

❌ **Don't blindly accept** - Always review AI-generated code

❌ **Don't make prompts too complex** - Break into smaller tasks

❌ **Don't ignore errors** - Ask AI to explain and fix issues

❌ **Don't skip testing** - Verify locally before pushing

Prompt Templates by Task

Website Structure

```
Create a [business type] website with:
- Home page with hero, features, testimonials, CTA
- About page with story and team
- Services page with [number] service cards
- Contact page with form and contact info
- Navigation and footer on all pages
- Responsive design
- [Color scheme] color palette
```

Component Creation

```
Create a [component name] component that:
- Accepts props: [list props]
- Renders: [describe output]
- Has states: [list states]
- Uses: [specific libraries if any]
- Matches style: [reference existing components]
```

API Integration

```
Create an API route at /api/[endpoint] that:
- Accepts [method] requests
- Validates [input data]
- Connects to [service/database]
- Returns [response format]
- Handles errors with [error format]
```

Styling Tasks

```
Apply these design updates:
- Color scheme: [colors]
- Typography: [font preferences]
- Spacing: [spacing scale]
```

```
- Components to update: [list]
- Breakpoints: [responsive requirements]
```

Bug Fixes

```
Fix this issue: [describe problem]
Error message: [paste error]
File: @[file path]
Expected behavior: [what should happen]
Current behavior: [what actually happens]
```

Troubleshooting with AI

Common Scenarios

Build Error - Missing Import

Error: `Module not found: Can't resolve '@/components/ui/button'`

Prompt:

```
Fix this import error in @app/page.tsx:
"Cannot find module '@/components/ui/button'"

Check if the file exists and fix the import path.
```

TypeScript Error

Error: `Type 'string' is not assignable to type 'number'`

Prompt:

```
Fix the TypeScript error in @[file]:
"Type 'string' is not assignable to type 'number'"

The error is on line [X]. Convert the value properly.
```

Styling Issue

Problem: Component not responsive on mobile

Prompt:

```
The hero section looks bad on mobile (under 768px).
Current issues: [describe]

Fix the responsive design using Tailwind breakpoints.
```

Deployment Issue

Problem: Vercel build fails

Prompt:

```
The Vercel deployment is failing with:
[paste build log]

Identify the issue and provide the fix.
```

Example: Complete Project Walkthrough

Project: Digital Agency Website

Step 1: Setup

Prompt:

```
Create a new Next.js project called "digital-agency" with:
- TypeScript
- Tailwind CSS
- shadcn/ui
- App Router
- Dark mode support
- Place it in ~/projects/
```

Step 2: Home Page

Prompt:

```
In the digital-agency project, create a home page with:
1. Navigation with logo and links (Home, Services, About, Contact)
2. Hero section:
   - Headline: "We Build Digital Experiences"
   - Subheadline about web development services
   - CTA button linking to /contact
   - Animated background
3. Features section with 3 cards:
   - Web Design
   - Development
   - Marketing
4. Testimonials slider with 3 quotes
5. Footer with contact info and social links
```

Step 3: Services Page

Prompt:

```
Create a services page at /services with:
- Page title and description
- 6 service cards in a 3x2 grid:
  1. Web Design (icon: Palette)
  2. Development (icon: Code)
  3. SEO (icon: Search)
  4. Marketing (icon: TrendingUp)
  5. Branding (icon: Brush)
  6. Support (icon: Headphones)
```

```
- Each card: icon, title, description, learn more link
- Consistent styling with home page features
```

Step 4: About Page

Prompt:

```
Create an about page with:
- Company story section with text and image
- Mission statement
- Team section:
  - 4 team members
  - Circular photos
  - Names and roles
  - Social media links
- Timeline of company milestones
- Values section with icons
```

Step 5: Contact Page

Prompt:

```
Create a contact page with:
- Contact form:
  - Name (required)
  - Email (required, validated)
  - Phone (optional)
  - Message (required, textarea)
  - Submit button
- Contact information sidebar:
  - Address
  - Phone
  - Email
  - Business hours
- Map placeholder (we'll add real map later)
- Form validation with error messages
- Success state after submission
```

Step 6: Polish

Prompt:

```
Polish the entire website:
1. Add smooth scroll behavior
2. Add page transition animations
3. Create a loading component
4. Add SEO meta tags to all pages
5. Ensure all links work correctly
6. Test responsive design on all pages
7. Optimize images with next/image
```

Step 7: Deploy

Prompt:

```
Help me deploy this digital-agency website:
1. Initialize git repository
2. Create GitHub repo and push
3. Deploy to Vercel with GitHub integration
4. Configure build settings
5. Set up custom domain (optional)

Provide the exact commands to run.
```

Cascade provides:

```
# Git setup
git init
git add .
git commit -m "Initial commit"
gh repo create digital-agency --public --source=. --push

# Vercel setup
vercel
# Follow prompts to link to GitHub and deploy
```

Key Takeaways

The Vibe-Coding Mindset

1. **Describe, Don't Write** - Use natural language to describe what you want
2. **Iterate Rapidly** - Make small prompts, review, refine
3. **Trust but Verify** - AI accelerates, but you guide quality
4. **Learn Continuously** - Ask AI to explain its choices
5. **Focus on Product** - Spend more time on user experience, less on syntax

AI Collaboration Levels

Level	Your Role	AI Role
Beginner	Describe features	Writes all code
Intermediate	Design architecture	Implements components
Advanced	Set patterns & review	Assists with implementation
Expert	Strategic decisions	Accelerates execution

Getting Started Today

1. **Install Windsurf** - Download from codeium.com
2. **Open a project** - Any existing or new folder
3. **Open Cascade** - Press `Ctrl/Cmd + L`
4. **Type your first prompt** - "Create a simple landing page"
5. **Iterate** - Keep refining with follow-up prompts

6. **Deploy** - Push to GitHub, deploy to Vercel

Resources

Prompt Libraries

- Keep a file of prompts that worked well
- Share prompts with your team
- Document which prompts produce best results

Learning Path

1. Start with simple component prompts
2. Progress to full page generation
3. Learn to debug with AI
4. Master the iteration cycle
5. Build complete projects independently

Community

- **Windsurf Discord** - Share prompts and tips
- **GitHub Discussions** - Next.js and Vercel communities
- **Twitter/X** - Follow #vibecoding for inspiration

Hands-On Exercises: Mastering the Windsurf Workflow

Complete these exercises to build muscle memory for common tasks. Each exercise includes step-by-step instructions and explains why the skill matters.

Exercise 1: Creating a Workspace for a New Project

Objective: Set up a proper workspace structure for a portfolio website project.

Steps:

1. Open Windsurf
2. Click `File` → `Open Folder`
3. Navigate to `Documents` and create a new folder named `portfolio-exercise`
4. Select the folder and click `Open`
5. Once loaded, click `File` → `Save Workspace As...`
6. Save as `portfolio-exercise.code-workspace` in `Documents/Workspaces/`
7. Verify the workspace opens correctly by closing and reopening Windsurf

Check Your Understanding:

- Where is the `.code-workspace` file stored?
- What happens when you double-click a workspace file?

Why This Matters: Workspaces preserve your entire development environment—open files, terminal history, and settings. Without workspaces, you'd spend 5-10 minutes recreating your setup every time you switch projects. With workspaces, you jump straight into coding.

Exercise 2: Accessing Previous Projects

Objective: Practice switching between multiple projects using workspace files.

Steps:

1. Create a second project folder: `Documents/blog-exercise`
2. Open it in Windsurf and save as `blog-exercise.code-workspace`
3. Close Windsurf completely
4. Navigate to `Documents/Workspaces/` in your file explorer
5. Double-click `portfolio-exercise.code-workspace`
6. Note: Portfolio project loads with all previous state
7. Switch projects: `File` → `Open Workspace from File...` → Select `blog-exercise.code-workspace`
8. Try the Recent menu: `File` → `Open Recent` → Switch back to portfolio

Check Your Understanding:

- How many recent workspaces does Windsurf remember?
- What's the fastest way to switch between two active projects?

Why This Matters: Developers often work on 3-5 projects simultaneously. Fast project switching means you can respond to client feedback on Project A, fix a bug on Project B, and resume feature work on Project C without losing context.

Exercise 3: Using the Terminal to Run Next.js Locally

Objective: Start a Next.js development server and view your site locally.

Prerequisites: Node.js installed (v18+)

Steps:

1. Open your `portfolio-exercise` workspace
2. Open Terminal: Press `Ctrl+`` or `Terminal` → `New Terminal`
3. Verify Node.js:

   ```
   node --version
   ```

4. Create Next.js project:

   ```
   npx create-next-app@latest .
   ```

5. Answer prompts:
   - TypeScript: Yes
   - ESLint: Yes
   - Tailwind CSS: Yes
   - `src` directory: Yes
   - App Router: Yes
   - Import alias: No
6. Wait for installation (2-5 minutes)
7. Start dev server:

   ```
   npm run dev
   ```

8. Open browser to `http://localhost:3000`
9. Make a small edit to `src/app/page.tsx`
10. Watch the browser auto-refresh
11. Stop server:

```
Ctrl+C
```

Check Your Understanding:

- What URL do you use to view your local site?
- What command stops the server?
- What happens when you save a file?

Common Terminal Commands:

```
npm run dev      # Start development server
npm run build    # Build for production
npm start        # Start production server
Ctrl+C           # Stop running server
```

Why This Matters: Local development lets you test changes instantly before showing anyone. The dev server provides hot reloading (instant updates), error overlays, and source maps for debugging. This is your safe sandbox—break things here, not on the live site.

Exercise 4: Understanding the Next.js Project Structure

Objective: Explore and understand the key files and folders in a Next.js project.

Steps:

1. With your portfolio project open, examine the Explorer panel
2. Open each folder and read the files:

Explore These Files:

Path	Purpose
`src/app/page.tsx`	Home page component
`src/app/layout.tsx`	Root layout (wraps all pages)
`src/app/globals.css`	Global CSS styles
`public/`	Static assets (images, fonts)
`components/`	Reusable React components
`next.config.js`	Next.js configuration
`package.json`	Dependencies and scripts
`tsconfig.json`	TypeScript settings

Hands-On Exploration:

1. Open `src/app/page.tsx` - this renders at `/`
2. Open `src/app/layout.tsx` - notice the `<html>` and `<body>` tags
3. Create a new folder: `src/app/about`
4. Create a file: `src/app/about/page.tsx`
5. Add this code:

```
export default function AboutPage() {
  return <h1>About Me</h1>
}
```

6. Visit `http://localhost:3000/about`

Check Your Understanding:

- Where do you put images you want to use?
- What file creates the `/about` route?
- What's the difference between `page.tsx` and `layout.tsx`?

Why This Matters: Understanding the project structure lets you navigate confidently. You'll know where to add pages, where to put components, and which files control behavior. This mental model is essential for effective AI prompting—you need to reference the right files.

Exercise 5: Setting Up .env.local and Understanding Environment Variables

Objective: Create and use environment variables for sensitive configuration.

Steps:

1. In your project root, create a file named `.env.local`
2. Add these variables:

```
# Public variables (visible in browser)
NEXT_PUBLIC_SITE_URL=http://localhost:3000
NEXT_PUBLIC_SITE_NAME=My Portfolio

# Private variables (server only)
DATABASE_URL=your-database-url
API_SECRET_KEY=your-secret-key
ADMIN_EMAIL=admin@example.com
```

3. Save the file
4. Open `src/app/page.tsx`
5. Add this to display the public variable:

```
export default function Home() {
  return (
    <main>
      <h1>Welcome to {process.env.NEXT_PUBLIC_SITE_NAME}</h1>
      <p>Site URL: {process.env.NEXT_PUBLIC_SITE_URL}</p>
    </main>
  )
}
```

6. Verify you can see "My Portfolio" on the page
7. Check `.gitignore` - verify `.env.local` is listed
8. Try to access `process.env.DATABASE_URL` in the component - it will be undefined (correct behavior)

Rules of Environment Variables:

- `NEXT_PUBLIC_` prefix = visible in browser
- No prefix = server-only, secure
- Never commit `.env.local` to Git (check `.gitignore`)
- Create `.env.example` with dummy values for team members

Check Your Understanding:

- Why use `NEXT_PUBLIC_` prefix?
- What happens if you commit `.env.local` to Git?
- Where do private variables work vs public variables?

Why This Matters: Environment variables separate configuration from code. You can have different settings for development (local database) vs production (live database) without changing code. More importantly, they keep secrets (API keys, passwords) out of your code repository where others might see them.

Real-World Usage:

```
# .env.local (never commit this)
NEXT_PUBLIC_STRIPE_PUBLISHABLE_KEY=pk_test_...
STRIPE_SECRET_KEY=sk_test_...
NEXTAUTH_SECRET=random-string
ADMIN_PASSWORD=secure-password
```

Exercise 6: Setting Up GitHub from Scratch

Objective: Create a GitHub account and understand the platform basics.

Steps:

1. Visit github.com
2. Click `Sign up` and complete registration
3. Verify your email address
4. Log in to GitHub
5. Click your profile picture (top right) → `Your repositories`
6. Click the green `New` button
7. Create a test repository:
    - Repository name: `learning-github`
    - Description: "My first GitHub repository"
    - Visibility: Public
    - Check `Add a README file`
    - Click `Create repository`
8. Explore the repository page:
    - Click on `README.md` to view it
    - Click the pencil icon to edit
    - Add a line: "I'm learning GitHub!"
    - Scroll down, add commit message: "Update README"

    - Click `Commit changes`
9. View the commit history by clicking the commit count

Key GitHub Concepts:

- **Repository**: Project folder with version history
- **README.md**: Project description shown on the main page
- **Commit**: Saved snapshot of changes
- **Public vs Private**: Who can see your code

Check Your Understanding:

- What's the difference between a repository and a folder?
- What does a README file do?
- Why write descriptive commit messages?

Why This Matters: GitHub is the world's code repository. It's where you store projects, collaborate with others, and showcase your work to employers. Without GitHub, your code lives only on your computer—lose that, lose everything. With GitHub, your code is backed up, shareable, and discoverable.

Exercise 7: Connecting a Windsurf Project to GitHub

Objective: Link your local Next.js project to a GitHub repository.

Prerequisites: Git installed, GitHub account created

Steps:

1. In Windsurf, open your `portfolio-exercise` project
2. Open Terminal (`Ctrl+``)
3. Initialize Git repository:

   ```
   git init
   ```

4. Check status:

   ```
   git status
   ```

   (You'll see many untracked files in red)
5. Create a `.gitignore` file if it doesn't exist:

   ```
   # Dependencies
   /node_modules
   /.pnp
   .pnp.js

   # Testing
   /coverage

   # Next.js
   /.next/
   /out/
   ```

```
# Production
/build

# Misc
.DS_Store
*.pem

# Debug
npm-debug.log*
yarn-debug.log*
yarn-error.log*

# Local env files
.env*.local
.env

# Vercel
.vercel

# TypeScript
*.tsbuildinfo
next-env.d.ts
```

6. Stage all files:

```
git add .
```

7. Make your first commit:

```
git commit -m "Initial commit: Next.js project setup"
```

8. Go to GitHub and create a new repository named `portfolio-exercise` (don't initialize with README)
9. Copy the repository URL (e.g., `https://github.com/yourusername/portfolio-exercise.git`)
10. Connect local to remote:

```
git remote add origin https://github.com/yourusername/portfolio-exercise.git
git branch -M main
git push -u origin main
```

11. Refresh your GitHub page - code is now there!

Daily Git Workflow:

```
git status                    # Check what's changed
git add .                     # Stage all changes
git commit -m "Description"   # Save with message
git push                      # Upload to GitHub
```

Check Your Understanding:

- What's the purpose of `git init`?

- Why do we need `.gitignore`?
- What does `git push` do?

Troubleshooting:

- **Error**: "fatal: not a git repository" → Run `git init` first
- **Error**: "Permission denied" → Check GitHub credentials
- **Error**: "Updates were rejected" → Run `git pull` first, then push

Why This Matters: Git is your safety net and collaboration tool. It tracks every change, so you can:

- Revert mistakes instantly
- See who changed what and when
- Work on multiple features simultaneously
- Collaborate without overwriting each other's work
- Deploy specific versions to production

Exercise 8: Connecting a GitHub Project to Vercel

Objective: Deploy your GitHub repository to Vercel for automatic hosting.

Prerequisites: GitHub repository with code, Vercel account

Steps:

1. Visit vercel.com
2. Click `Sign Up` → Choose `Continue with GitHub`
3. Authorize Vercel to access your GitHub account
4. On Vercel dashboard, click `Add New...` → `Project`
5. Find your `portfolio-exercise` repository and click `Import`
6. Configure project:
    - Framework Preset: Next.js (auto-detected)
    - Root Directory: `./`
    - Build Command: `next build` (default)
    - Output Directory: `.next` (default)
7. Click `Deploy`
8. Wait for build (2-3 minutes)
9. Click the URL (e.g., `portfolio-exercise-xxx.vercel.app`)
10. Your site is live on the internet!

Understanding the Integration:

- Vercel watches your GitHub repository
- Every push to `main` triggers automatic redeployment
- Preview deployments are created for pull requests
- Your `.env.local` variables need to be added in Vercel dashboard

Setting Environment Variables on Vercel:

1. Go to Vercel Dashboard → Select your project
2. Click `Settings` tab → `Environment Variables`
3. Add each variable from your `.env.local`:
    - Name: `NEXT_PUBLIC_SITE_NAME`
    - Value: `My Portfolio`

4. Click `Save`
5. Redeploy for changes to take effect

Check Your Understanding:

- What happens when you push changes to GitHub?
- Where do you set production environment variables?
- How long does deployment typically take?

Why This Matters: Vercel turns Git pushes into live websites automatically. No manual uploads, no FTP, no server configuration. This enables:

- Continuous deployment: Code changes → Live site in minutes
- Preview URLs: Test changes before merging
- Rollbacks: Revert to previous versions instantly
- Global CDN: Your site loads fast worldwide
- HTTPS: Secure connections by default

Exercise 9: Watching GitHub Push Updates to Vercel

Objective: Make a code change and watch it automatically deploy.

Steps:

1. Ensure your project is deployed on Vercel from Exercise 8
2. In Windsurf, open `src/app/page.tsx`
3. Make a visible change:

```
export default function Home() {
  return (
    <main className="p-8">
      <h1 className="text-3xl font-bold text-blue-600">
        My Portfolio - Updated!
      </h1>
      <p>This change was pushed to GitHub and auto-deployed to Vercel.</p>
    </main>
  )
}
```

4. Save the file
5. Open Terminal
6. Stage and commit:

```
git add .
git commit -m "Update homepage with new heading"
```

7. Push to GitHub:

```
git push
```

8. Watch the magic happen:
    - Go to your GitHub repository

    - Click `Actions` tab (see commit being processed)
    - Go to Vercel dashboard
    - Click on your project
    - Watch the build progress indicator
9. Once build completes (green checkmark), visit your live URL
10. See your changes live on the internet!

Understanding the Pipeline:

```
Local Code Change
    ↓
git commit & git push
    ↓
GitHub receives commit
    ↓
Vercel detects change
    ↓
Build starts automatically
    ↓
Build completes
    ↓
Site updated globally
```

Check Your Understanding:

- How long did the deployment take?
- Where can you see the build status?
- What happens if the build fails?

Exploring Build Logs:

1. In Vercel dashboard, click on a deployment
2. Click `Build Logs`
3. See every step: installing dependencies, building, optimizing
4. If something fails, the error appears here

Why This Matters: This workflow—code, commit, push, auto-deploy—is the modern standard. It eliminates:

- Manual file uploads
- "It works on my machine" problems
- Fear of deploying (easy rollbacks)
- Downtime during updates

You can make a fix, push it, and have it live in 2 minutes. Clients see updates instantly. Bugs get fixed fast. This velocity is what makes modern web development so powerful.

Exercise Summary: Why Each Step Matters

Exercise	Skill	Why It Matters
1. Workspaces	Project organization	Saves 5-10 minutes per context switch, preserves mental state
2. Accessing Projects	Navigation	Enables managing multiple projects without friction

3. Local Dev Server	Development environment	Safe sandbox for experimentation with instant feedback
4. Project Structure	Code navigation	Know where everything lives; reference files correctly in prompts
5. Environment Variables	Security & configuration	Keep secrets safe, deploy to multiple environments
6. GitHub Setup	Version control	Backup, collaboration, history, portfolio showcase
7. Git Integration	Code management	Track changes, collaborate, deploy specific versions
8. Vercel Connection	Deployment	Turn code into live websites automatically
9. CI/CD Pipeline	Modern workflow	Ship changes fast, iterate rapidly, deploy confidently

The Big Picture: These exercises teach the complete modern development cycle:

1. **Plan** → Define what you're building (Exercise 5 planning template)
2. **Develop** → Build locally with instant feedback (Exercises 1-4)
3. **Secure** → Manage configuration safely (Exercise 5)
4. **Version** → Track changes with Git (Exercise 7)
5. **Store** → Back up on GitHub (Exercise 6)
6. **Deploy** → Go live with Vercel (Exercise 8)
7. **Iterate** → Rapid improvement cycle (Exercise 9)

Master these 9 exercises, and you have the foundational skills to build and deploy any web application using AI assistance.

Appendix: Understanding Localhost and Development Servers

What is Localhost:3000?

Localhost refers to your own computer. When you type `localhost` in a browser, you're telling it to look for a website on your machine rather than on the internet.

The :3000 part is a "port number." Think of it like a specific door on your computer:

- Port 3000 is the default door Next.js uses
- Other common ports: 80 (standard web), 443 (secure web), 8080 (alternative development)
- Multiple services can run on different ports simultaneously

In simple terms: `http://localhost:3000` = "Show me the website running on my computer at door 3000"

What is a Development Server?

A development server is a program that:

1. Serves your website files to your browser
2. Watches for file changes and auto-refreshes
3. Shows errors and debugging information
4. Compiles code (TypeScript, JSX, etc.) on the fly

Why use a dev server instead of just opening files?

Just Opening Files	Using Dev Server
File:// protocol (limited)	http://localhost (full browser features)
Manual refresh after changes	Auto-refresh on save
No error messages	Detailed error overlays
Can't use modern features	Hot module replacement, routing, etc.

How to Run the Development Server

Starting the Server

1. **Open Terminal** in Windsurf (`Ctrl+``)
2. **Navigate to your project folder** (if not already there):

```
cd ~/Documents/my-project
```

3. **Install dependencies** (first time only):

```
npm install
```

4. **Start the dev server**:

```
npm run dev
```

5. **Wait for the message**:

```
▲ Next.js 14.x
- Local:        http://localhost:3000
- Environments: .env.local
- Experiments:  (empty)
```

6. **Open browser** and go to `http://localhost:3000`

Common Commands

```
# Start development server
npm run dev

# Build for production
npm run build

# Start production server (after building)
npm start

# Run linting
npm run lint
```

```
# Stop any running server
Ctrl+C
```

What is Local Development?

Local development means building and testing on your own computer before showing anyone else.

The Local Development Workflow:

```
1. Make changes in Windsurf
        ↓
2. Save files (Ctrl+S)
        ↓
3. Dev server detects changes
        ↓
4. Browser auto-refreshes
        ↓
5. See results instantly
        ↓
6. Repeat until satisfied
        ↓
7. Commit and push to deploy
```

Benefits of Local Development:

- **Speed**: Changes appear in milliseconds, not minutes
- **Safety**: Break things privately, not on the live site
- **Debugging**: Full error messages and source maps
- **Offline work**: No internet required to code
- **Experimentation**: Try radical changes without consequences

Troubleshooting Local Development

Problem: "localhost:3000 refused to connect"

Causes & Solutions:

1. **Server not running**
    - Check terminal - is `npm run dev` still running?
    - If not, restart it
2. **Wrong port**
    - Check what port the server actually started on
    - Look for message: `- Local: http://localhost:3000`
    - If it says 3001, use that instead
3. **Port already in use**
    - Error: "Port 3000 is already in use"
    - Solution: Find and close the other program, or use a different port:

      ```
      npm run dev -- --port 3001
      ```

Problem: Changes not showing in browser

Solutions:

1. Check you saved the file (Ctrl+S)
2. Hard refresh browser: `Ctrl+Shift+R` (Windows) or `Cmd+Shift+R` (Mac)
3. Check browser console (F12) for errors
4. Restart dev server:

```
Ctrl+C  # Stop server
npm run dev  # Start again
```

Problem: Terminal shows errors

Read the error message carefully:

- "Module not found" → Missing import or package
- "Syntax error" → Typo in code
- "Cannot find name" → Missing TypeScript type
- "EACCES permission denied" → File permission issue

Problem: npm install fails

Solutions:

1. Check Node.js version:

```
node --version  # Should be v18 or higher
```

2. Clear npm cache:

```
npm cache clean --force
```

3. Delete node_modules and reinstall:

```
rm -rf node_modules
rm package-lock.json
npm install
```

Understanding the URL Bar

When developing locally, you'll see these URLs:

URL	Meaning
`http://localhost:3000`	Your local dev server
`http://localhost:3000/about`	The /about page route
`http://localhost:3000/api/hello`	API endpoint
`file:///C:/Users/...`	Just opening a file directly (avoid this)

Production vs Development

Feature	Development (`npm run dev`)	Production (`npm run build` + `npm start`)
Speed	Fast compilation	Optimized, minified code
Errors	Detailed error pages	Generic error pages
Hot Reload	Yes	No
Source Maps	Full	None (or minimal)
Performance	Unoptimized	Highly optimized
Use For	Coding	Deploying

Quick Reference: Local Development Checklist

Before you start coding:

- [ ] Terminal is open
- [ ] In the correct project folder
- [ ] Ran `npm install` (if first time)
- [ ] Dev server running (`npm run dev`)
- [ ] Browser open to `http://localhost:3000`
- [ ] Can see the site loading

When you're done coding:

- [ ] Stop dev server (Ctrl+C)
- [ ] Commit changes: `git add . && git commit -m "message"`
- [ ] Push to GitHub: `git push`
- [ ] Verify deployment on Vercel

Conclusion

Vibe-coding with Windsurf, GitHub, and Vercel transforms web development from manual coding to collaborative creation. By mastering AI prompting, you can:

- **Build faster** - Prototypes in hours, not days
- **Explore more** - Try ideas without technical barriers
- **Focus on value** - Spend time on user experience
- **Ship confidently** - AI-assisted code + automated deployment

Your new workflow:

```
Prompt → AI Generates → You Review → Iterate → Push → Auto-Deploy
```

Start your first AI-powered project today!

Document Version: 3.1 - With Workspace Optimization for Learning **Last Updated:** March 2025