AI Tools

Technical Guide

Prepared by

Kadharmoideen Fadurudeen

In this guide

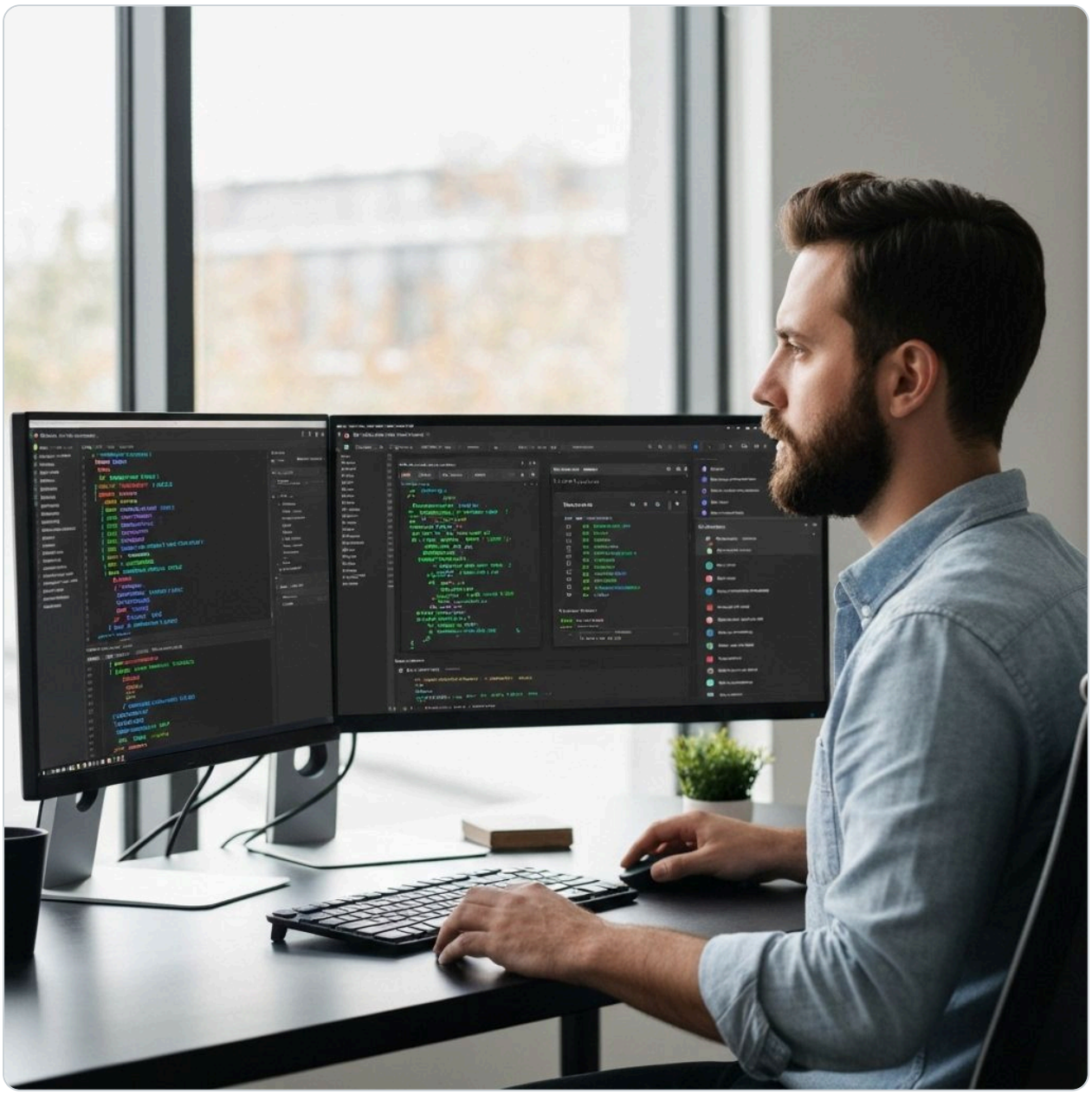
1 article · ~15 min total reading

- **How to Make Best Use of Cursor AI for Development**

AI Tools October 12, 2025 15 min read

How to Make Best Use of Cursor AI for Development

Discover how Cursor AI is revolutionizing software development with intelligent code completion, natural language commands, and AI-powered refactoring. Learn practical tips and real-world use cases to boost your productivity.



As a Lead Engineer with 19+ years of experience, I've witnessed countless tools promising to revolutionize development. Cursor AI is one of the few that actually delivers on that promise. Built on top of VS Code, Cursor combines the familiarity of your favorite editor with the power of advanced AI models to create a truly transformative coding experience.

What Makes Cursor AI Special?

Unlike traditional code completion tools, Cursor AI understands context across your entire codebase. It's not just autocomplete on steroids—it's an intelligent pair programmer that can understand your intent, suggest architectural improvements, and even write entire functions based on natural language descriptions.

Key Features

- **Codebase-aware completions:** Understands your entire project structure
- **Natural language commands:** Write code by describing what you want
- **Intelligent refactoring:** Suggest and implement code improvements
- **Multi-file editing:** Make changes across multiple files simultaneously

Practical Use Cases

1. Rapid Prototyping

One of my favorite use cases is rapid prototyping. Instead of writing boilerplate code, I describe what I need in natural language:

```
// Prompt to Cursor:
"Create a React component that fetches user data from an API, displays it in a table with sorting and filtering, and includes error handling with loading states"
```

Cursor generates a complete, production-ready component with proper TypeScript types, error boundaries, and even unit tests if you ask for them. Here's what the AI-assisted coding experience looks like:

UserTable.tsx

```tsx
importReactfrom'react'
import{ useState, useEffect }from'react'

exportfunctionUserTable() {
  const [users, setUsers] = useState([]);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
  // AI Suggestion: Fetch users with error handling
  const fetchUsers = async () => {
    try {
      const response = await fetch('/api/users');
      const data = await response.json();
      setUsers(data);
    } catch (error) {
      console.error('Failed to fetch users', error);
    } finally {
      setLoading(false);
    }
  };
```

Tab to accept

Cursor AI suggesting complete code blocks with error handling

More Rapid Prototyping Prompts:

"Build a responsive navbar with mobile menu, dark mode toggle, and smooth scroll to sections"

"Create a form with validation using React Hook Form and Zod schema, including email, password strength indicator, and submit handling"

"Generate a dashboard layout with sidebar navigation, header with user menu, and responsive grid for cards"

```
"Build a search component with debouncing, autocomplete suggestions,
and keyboard navigation"
```

2. Code Refactoring

Legacy code refactoring becomes significantly easier. Cursor can analyze your code and suggest improvements:

- Convert class components to functional components with hooks
- Migrate from JavaScript to TypeScript with proper type definitions
- Optimize performance by identifying bottlenecks
- Apply design patterns and best practices

Example Refactoring Prompt:

```
"Refactor this class component to a functional component with hooks,
extract custom hooks for data fetching, and add proper TypeScript
types"
```

Before:

```
class UserProfile extends React.Component {
  constructor(props) {
    super(props);
    this.state = { user: null, loading: true };
  }

  componentDidMount() {
    fetch('/api/user')
      .then(res => res.json())
      .then(user => this.setState({ user, loading: false }));
  }

  render() {
    return <div>{this.state.user?.name}</div>;
  }
}
```

After (Cursor AI refactored):

```
interface User {
  name: string;
  email: string;
}

function useUser() {
  const [user, setUser] = useState<User | null>(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    fetch('/api/user')
      .then(res => res.json())
      .then(data => {
        setUser(data);
        setLoading(false);
      });
  }, []);

  return { user, loading };
}

export function UserProfile() {
  const { user, loading } = useUser();

  if (loading) return <Skeleton />;
  return <div>{user?.name}</div>;
}
```

3. Documentation Generation

Writing documentation is often tedious, but Cursor makes it effortless. Select a function or class, and ask Cursor to generate comprehensive documentation:

```
// Before:
function processUserData(data, options) {
  // Complex logic here
}

// After Cursor AI:
```

```
/**
 * Processes user data with configurable options
 * @param {Object} data - The user data object
 * @param {Object} options - Processing options
 * @param {boolean} options.validate - Whether to validate data
 * @param {string} options.format - Output format (json|xml)
 * @returns {Promise<ProcessedData>} Processed user data
 * @throws {ValidationError} If data validation fails
 */
function processUserData(data, options) {
  // Complex logic here
}
```

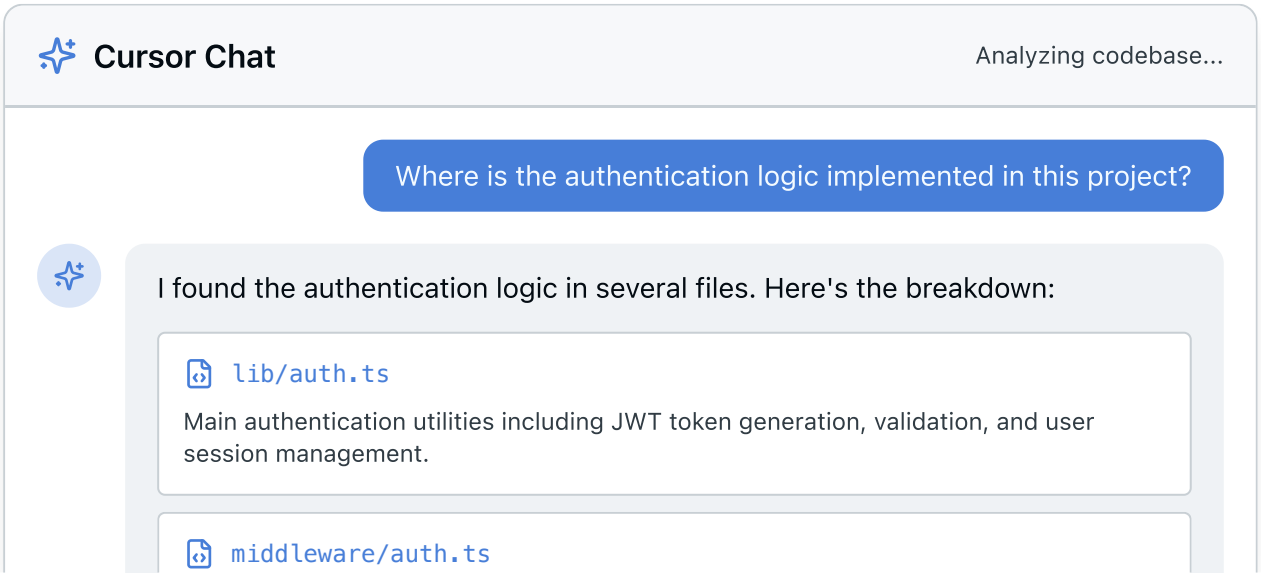
Advanced Techniques

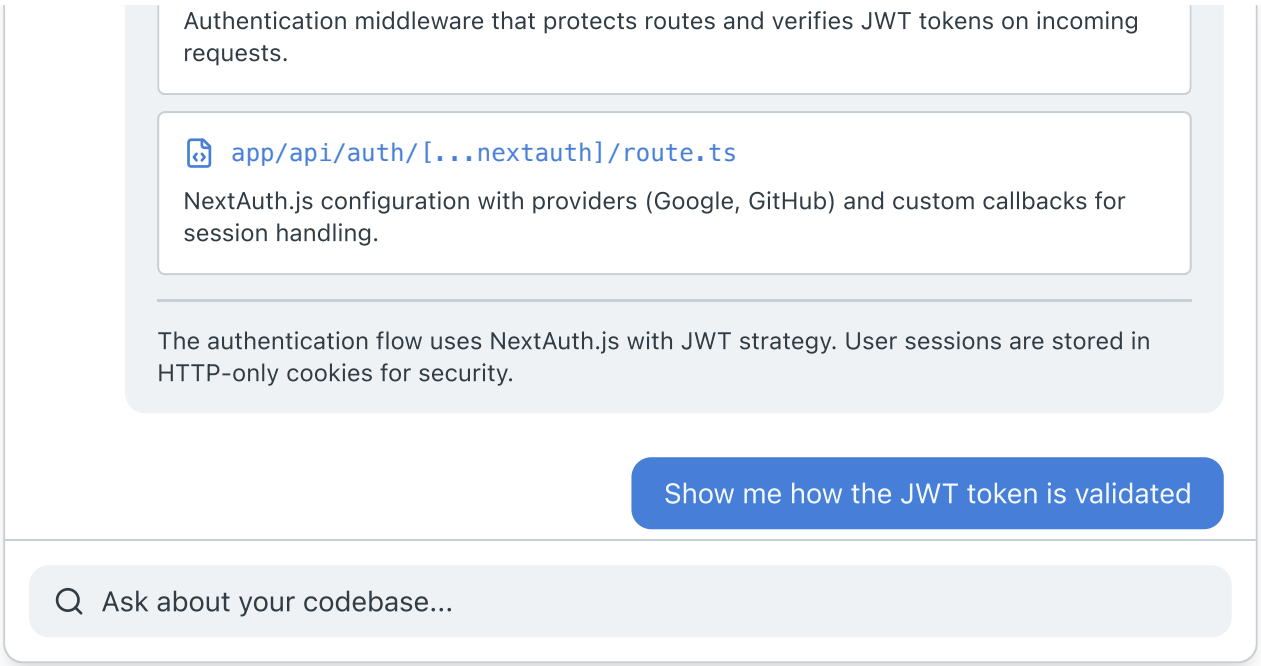
Codebase Chat

One of Cursor's most powerful features is the ability to chat with your codebase. Ask questions like:

- "Where is the authentication logic implemented?"
- "How does the payment processing flow work?"
- "What are all the API endpoints in this project?"
- "Find all components that use the useAuth hook"

Cursor analyzes your entire codebase and provides accurate, contextual answers with file references. Here's what the chat interface looks like:





Cursor Chat analyzing codebase and providing file references

Advanced Codebase Chat Prompts:

```
"Explain the data flow from the API to the UI in the user dashboard"

"Find all places where we're making database queries and list the tables being accessed"

"Show me all the environment variables used in this project and where they're referenced"

"Identify potential security vulnerabilities in the authentication flow"

"List all React components that make API calls and what endpoints they use"
```

Multi-File Edits

When refactoring affects multiple files, Cursor can make coordinated changes across your codebase:

```
// Example command:
"Rename the 'getUserData' function to 'fetchUserProfile' across all files and update all imports"
```

Cursor identifies all occurrences, updates function definitions, call sites, and import statements automatically.

Real-World Examples

Building a REST API

Recently, I used Cursor to build a complete REST API for a microservice:

What I asked Cursor:

1. "Create an Express.js API with TypeScript for managing user profiles"
2. "Add CRUD endpoints with proper validation using Zod"
3. "Implement JWT authentication middleware"
4. "Add error handling and logging with Winston"
5. "Generate OpenAPI documentation"

In under 30 minutes, I had a production-ready API with proper error handling, validation, and documentation. What would have taken hours of boilerplate writing was done in minutes.

Migrating to a New Framework

When migrating a React application to Next.js 14 with the App Router, Cursor helped me:

- Convert React Router routes to Next.js file-based routing
- Transform client components to server components where appropriate
- Update data fetching patterns to use Next.js conventions
- Implement proper loading and error states

Advanced Workflow Examples

Building a Feature End-to-End

Here's a real example of building a complete feature using Cursor AI:

Task: Add user profile editing functionality

> **Step 1: Database Schema**
> `"Create a Prisma schema for user profiles with fields: bio, avatar, location, website, social links"`

> **Step 2: API Endpoint**
> `"Create a Next.js API route for updating user profiles with validation, authentication check, and error handling"`

> **Step 3: Frontend Form**
> `"Build a profile edit form with React Hook Form, Zod validation, image upload with preview, and optimistic updates"`

> **Step 4: Tests**
> `"Generate Jest tests for the API endpoint covering success cases, validation errors, and authentication failures"`

Result: Complete feature implemented in under 20 minutes with proper validation, error handling, and tests.

Debugging Complex Issues

Cursor excels at helping debug tricky issues. Here's my workflow:

1. **Describe the bug:**
   `"Users are getting logged out randomly. The session seems to expire even though they're actively using the app"`
2. **Ask for analysis:**
   `"Analyze the authentication flow and identify potential causes for premature session expiration"`
3. **Get specific fixes:**
   `"Show me how to implement session refresh logic that extends the session on user activity"`

Best Practices and Tips

1. Be Specific with Context

The more context you provide, the better Cursor's suggestions. Instead of "add error handling," try:

```
"Add comprehensive error handling with try-catch blocks, log errors to
our Winston logger, and return appropriate HTTP status codes with
user-friendly error messages"
```

2. Use Cursor for Code Reviews

Before committing code, ask Cursor to review it:

- "Review this code for potential bugs and security issues"
- "Suggest performance optimizations"
- "Check if this follows our coding standards"

3. Leverage Cursor for Testing

Generate comprehensive test suites effortlessly:

```
// Command:
"Generate Jest unit tests for this component with test cases for all
props, user interactions, and edge cases"
```

4. Create Custom Commands

Set up custom commands for repetitive tasks:

- "Generate a new React component with TypeScript, props interface, and basic styling"
- "Create an API endpoint with validation, error handling, and tests"
- "Add comprehensive JSDoc comments to this file"

Pro Tips for Maximum Productivity

1. Context is King

Always provide context about your project structure, tech stack, and coding conventions:

```
"We're using Next.js 14 with App Router, TypeScript, Tailwind CSS,
and Prisma. Follow our convention of using server components by
default and only use 'use client' when necessary for interactivity"
```

2. Iterative Refinement

Don't expect perfection on the first try. Refine the output:

```
→ "Make this more performant by memoizing expensive calculations"
→ "Add loading skeletons instead of a simple spinner"
→ "Improve error messages to be more user-friendly"
```

3. Learn from the AI

When Cursor suggests something unfamiliar, ask for explanation:

```
"Explain why you used useMemo here and what performance benefit it
provides"
```

4. Combine with Other Tools

Use Cursor alongside your existing workflow:

- GitHub Copilot for inline suggestions
- ESLint/Prettier for code formatting
- TypeScript for type safety
- Traditional debugging tools for complex issues

Code Review and Optimization

Use Cursor as your code review partner:

Comprehensive Review Prompt:

```
"Review this component for: 1) Performance issues and unnecessary re-renders, 2) Accessibility problems, 3) Security vulnerabilities, 4) Code style and best practices, 5) Potential edge cases not handled"
```

Cursor's typical response includes:

- ✓ Specific line numbers with issues
- ✓ Explanation of why it's a problem
- ✓ Suggested fix with code examples
- ✓ Alternative approaches to consider

Productivity Gains

After using Cursor AI for several months, I've measured significant productivity improvements:

Measured Impact:

Boilerplate code generation

70% faster

Code refactoring tasks

60% faster

Documentation writing

80% faster

Bug fixing time

40% reduction

Limitations and Considerations

While Cursor AI is powerful, it's important to understand its limitations:

- **Always review generated code:** AI can make mistakes or miss edge cases

- **Security considerations:**Be cautious with sensitive code and credentials
- **Learning curve:**Takes time to learn effective prompting techniques
- **Context limits:**Very large codebases may exceed context windows

Conclusion

Cursor AI represents a fundamental shift in how we write code. It's not about replacing developers—it's about augmenting our capabilities and eliminating tedious tasks so we can focus on solving complex problems and building innovative solutions.

As someone who's been coding for nearly two decades, I can confidently say that Cursor AI is one of the most impactful tools I've adopted. It has transformed my daily workflow, increased my productivity by an estimated 50-70%, and made coding more enjoyable by handling the repetitive parts.

The key to success with Cursor is treating it as a collaborative partner, not a magic wand. Provide clear context, iterate on suggestions, and always review the generated code. With practice, you'll develop an intuition for crafting effective prompts and integrating AI assistance seamlessly into your workflow.

Ready to get started?

Download Cursor AI from cursor.sh and experience the future of AI-powered development today. Start with simple prompts, experiment with different approaches, and watch your productivity soar.