# Cursor AI – Developer IDE Playbook

## Introduction

Cursor AI is a powerful AI-native code editor designed to streamline development workflows using LLMs. It augments the traditional coding experience by integrating AI assistance directly into the IDE, enabling developers to write, debug, and refactor code faster and more intuitively.  
  
This playbook describes how to install, configure, and use Cursor AI effectively within a software development team. It serves as both onboarding documentation and best-practices reference.

## Installation

### Prerequisites

* OS: Windows 10/11 64‑bit, macOS 10.15+, or modern Linux (Ubuntu 20.04+)
* RAM: ≥ 4 GB (8 GB+ recommended)
* Disk: ≥ 2 GB free
* Internet: Broadband with ≥ 10 Mbps

### Download & Setup

1. Download Cursor AI from https://www.cursor.so

2. Install using the appropriate installer for your platform.

3. Open the application and sign in using GitHub or Google.

4. Configure workspace folders and indexing preferences.

## AI Capabilities Overview

**AI Chat**

- Ask questions about the codebase.  
- Works like ChatGPT but with full project context.  
- Example: “What does this function do?” or “Generate unit tests for this file.”

**Inline Code Fix**

- Right-click on a line or block of code to get an AI fix.  
- Useful for explaining errors, fixing bugs, and improving performance.  
- Suggestions appear inline and can be reviewed before applying.

**Refactor Suggestions**

- Highlight a block of code, then right-click for AI-based refactoring.  
- Supports splitting functions, renaming variables, and simplifying logic.  
- Great for cleaning up legacy or complex code.

**AI File Search**

- Use AI to search the codebase semantically, not just by keyword.  
- Find related files, concepts, or functions faster than with standard search.  
- Helpful when onboarding or debugging large codebases.

**Generate Unit Tests**

- Automatically generate tests for selected functions or files.  
- Supports frameworks like Jest, PyTest, etc., depending on language.  
- Can be a huge time saver for test-driven development.

**Multi-file Context Awareness**

- Cursor understands your entire project, not just open files.  
- Great for tasks like renaming across files, updating related logic, or tracing dependencies.

# Reindexing the Project

If the AI isn’t recognizing recent changes, use the Reindex option in Cursor to update its internal project map. This ensures that all files, symbols, and changes are properly reflected in the AI's context.

# Automating Routine Tasks

Leverage Cursor’s built-in task automation by combining AI with command prompts. You can automate doc generation, boilerplate writing, test creation, and more using saved prompts or macros.

# Cursor Directory

Explore and contribute to prebuilt Cursor AI configurations at https://cursor.directory/. This community-curated resource helps you get started with `.cursorrules` and context configurations tailored to your tech stack.

# Tips from the Community

Insights from community guides (like dev.to articles) offer practical usage:

* Composer Feature: Modify and test multiple files together with the Composer panel (Floating, Sidebar, Expanded).
* Markdown as Context: Include markdown files with instructions to guide AI across the project.
* Clean Working Directory: Keep changes committed to avoid confusion in AI-generated diffs.  
  Documentation Links: Include links in your prompts to steer AI toward the latest practices.
* Voice-to-Text Tools: Use tools like Wispr Flow for quick prompt creation via voice.  
  Model Suggestions: Default to Claude 3.5 Sonnet; o1-mini can be better for reasoning-heavy tasks.

## Configuration

## Root File Configuration

Ensure your project's root file is clearly defined and properly configured. Cursor uses the root to understand file structure, scope indexing, and resolve dependencies correctly. This is especially important for monorepos and multi-package workspaces.

# .cursorignore File

The `.cursorignore` file allows you to exclude files or directories from AI indexing. This is useful to ignore build folders, dependencies, or irrelevant data that may pollute context. Use it like a `.gitignore` to keep AI responses clean and focused.

# .cursorrules File Configuration

The `.cursorrules` file defines project-specific guidance and instructions for Cursor AI. It helps standardize how AI behaves, ensuring code consistency and context awareness.  
  
Benefits include:  
- Embedding coding guidelines  
- Highlighting legacy patterns  
- Enforcing custom best practices  
  
This file is version-controlled and located at the root of the project.

### Settings Overview

The following JSON illustrates underlying settings concepts. Cursor’s UI manages these options—there is no direct config file.

{  
 "ai.provider": "OpenAI",  
 "ai.model": "gpt-4o",  
 "enableContextualAwareness": true,  
 "enableInlineSuggestions": true,  
 "showChangeDiffs": true  
}

### Options:

|  |  |  |
| --- | --- | --- |
| Option | Type | Description |
| ai.provider | string | e.g. "OpenAI" or "Anthropic" |
| ai.model | string | LLM model to use (gpt-4o, claude-3, etc.) |
| enableContextualAwareness | bool | Whether to scope completions to open files |
| enableInlineSuggestions | bool | Enables ghost text in the editor |
| showChangeDiffs | bool | Highlights what AI modified in the file |

## Key Workflows

### 1. Writing Code with AI Assistance

- Start typing in a file; suggestions will appear automatically.

- Press Tab to accept full suggestions.

- For more advanced instructions, type '// ask AI:' followed by a prompt.

// ask AI: Write a function to debounce an input handler in TypeScript

### 2. Refactoring Code

- Select a code block → Right click → Ask AI → Refactor

- Options include: rename, extract function, optimize logic, comment generation

### 3. Searching Codebase

Use Cmd+K (Mac) or Ctrl+K (Windows) to bring up the AI Command Bar:

"Where is the user authentication handled?"

### 4. Fixing Bugs

- Select code with an error → Ask AI: Why is this breaking?

- AI suggests both explanation and fixes (optionally with tests).

- You can apply the fix with 1-click, or manually edit with preview.

### 5. Documenting Code

Use Cmd+Shift+D or Right Click → Ask AI → Generate Docs

function calculateInterest(principal, rate, time) {  
 return (principal \* rate \* time) / 100;  
}

AI-generated doc:

/\*\*  
 \* Calculates the simple interest on a principal amount.  
 \* @param principal - The initial amount of money  
 \* @param rate - The annual interest rate (in %)  
 \* @param time - The time the money is invested for (in years)  
 \* @returns The simple interest  
 \*/

## Best Practices

### ✅ Do

- Use comments as prompts to guide AI clearly

- Regularly review AI-generated code

- Fine-tune project-wide settings for your LLM provider

- Use diff view before applying AI refactors

### ❌ Don’t

- Blindly trust AI with security-critical code

- Use AI to generate sensitive API keys or passwords

- Skip human review before merging AI changes

## Advanced Configuration

### .cursorconfig.json

Project-specific overrides can be added via:

{  
 "disableTelemetry": true,  
 "defaultModel": "gpt-4o",  
 "excludedDirs": ["node\_modules", "dist"],  
 "maxTokens": 2048  
}

## Frequently Asked Questions (FAQ)

Q: Can I use my own OpenAI API key?  
A: Yes. Go to Settings → API Access and paste your key.

Q: How is context managed across files?  
A: Cursor builds a semantic tree of your project and feeds only relevant nodes into the prompt window.

Q: Is it secure for enterprise use?  
A: Yes, enterprise versions offer offline LLM options and customizable data privacy configurations.

## Sample AI Prompts

// ask AI: Convert this to async/await style  
// ask AI: Add JSDoc to all exported functions  
// ask AI: Optimize this for performance  
// ask AI: What's wrong with this useEffect hook?