

Cursor 101 Demo Flow (with Screenshots)

An introductory walkthrough of Cursor's core features for new users.

Prerequisites

Before starting the demo:

1. **Start the server:** Run `/start-demo` to launch the development server and open the blog app in the browser
2. **Reset workspace:** Use `/reset` between sections to ensure a clean state for deterministic results

 **Deep Links:** Throughout this demo, clickable links like [Example Link] open a “Create chat with prompt” dialog. Click “**Create Chat**” to start the agent with the pre-filled prompt.

1. Layout Walkthrough

Cursor is an IDE (Integrated Development Environment) forked from VS Code, now fully built out with AI-native features.

Understanding the layout helps you navigate efficiently and customize your workspace for different tasks.

Overview

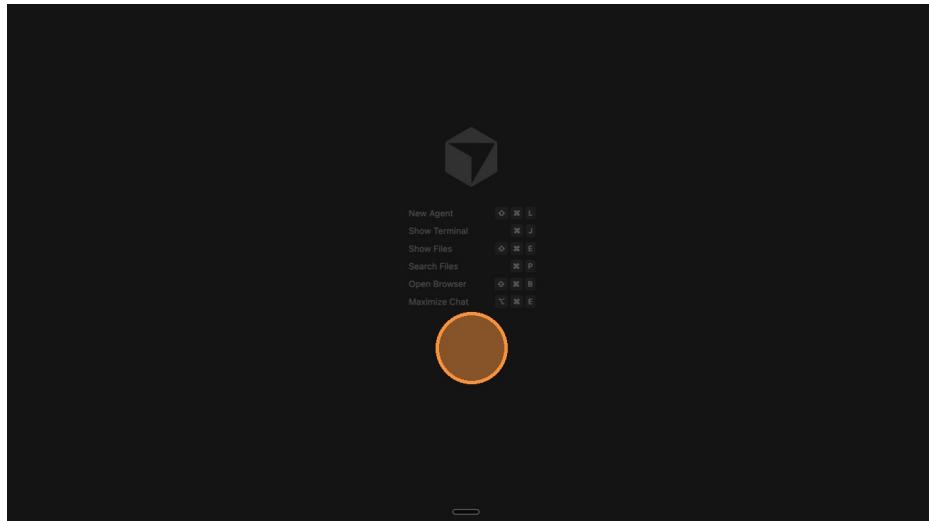
Cursor extends VS Code with AI-powered panels and features. The main areas include:

- **File Explorer** (left sidebar) - Navigate your project files
- **Editor** (center) - View and edit code
- **Terminal** (bottom panel) - Run commands
- **Agent Panel** (right sidebar) - AI chat and agent interactions

Demo

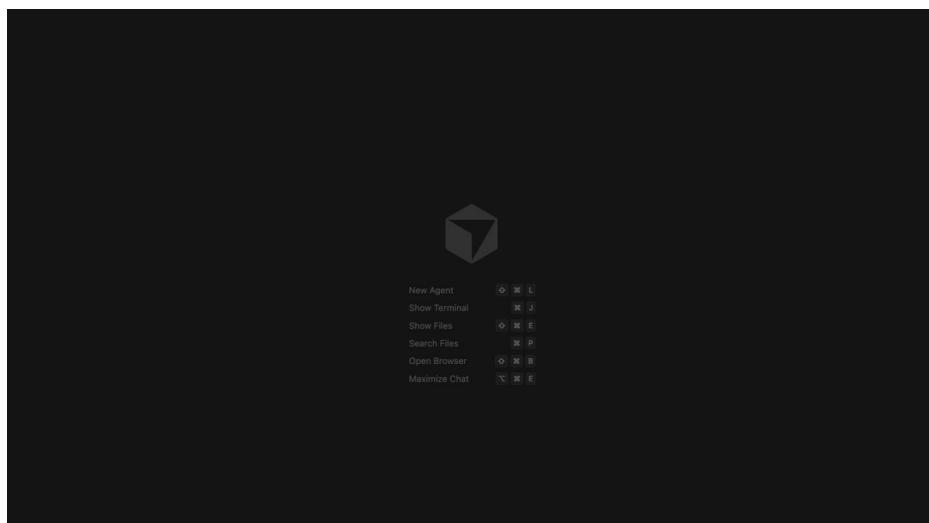
Opening the Layout

When you first open Cursor, you'll see an empty editor:



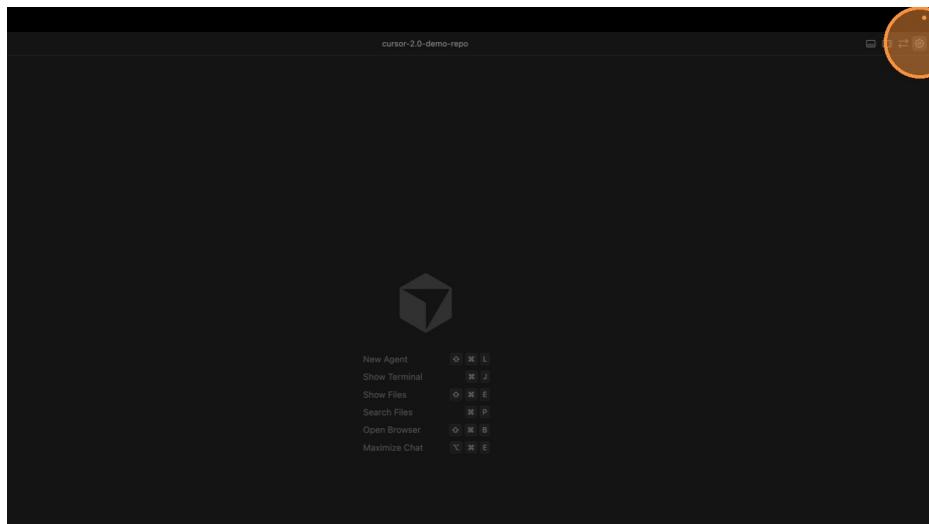
Empty editor view

Click “Change Layout” to customize your view:



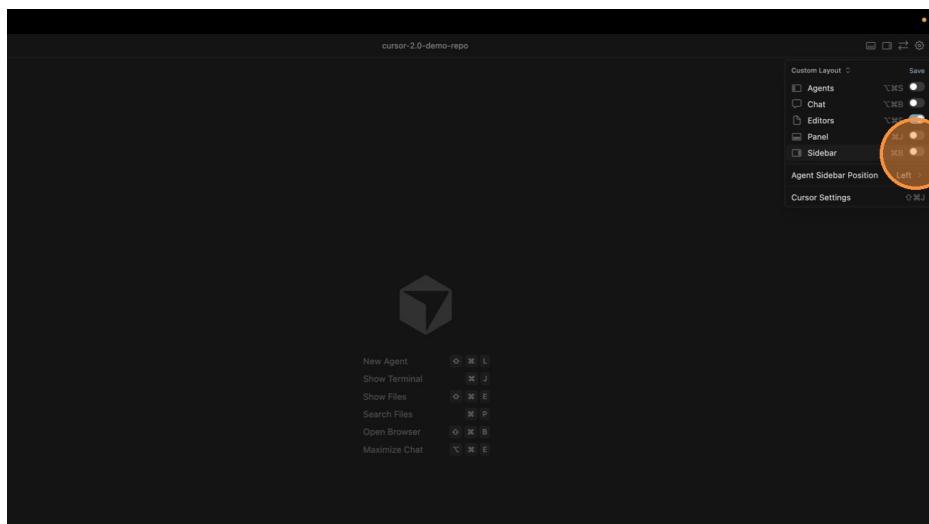
Change layout button

The layout options appear—Cursor is a fork of VS Code, now fully AI-native:



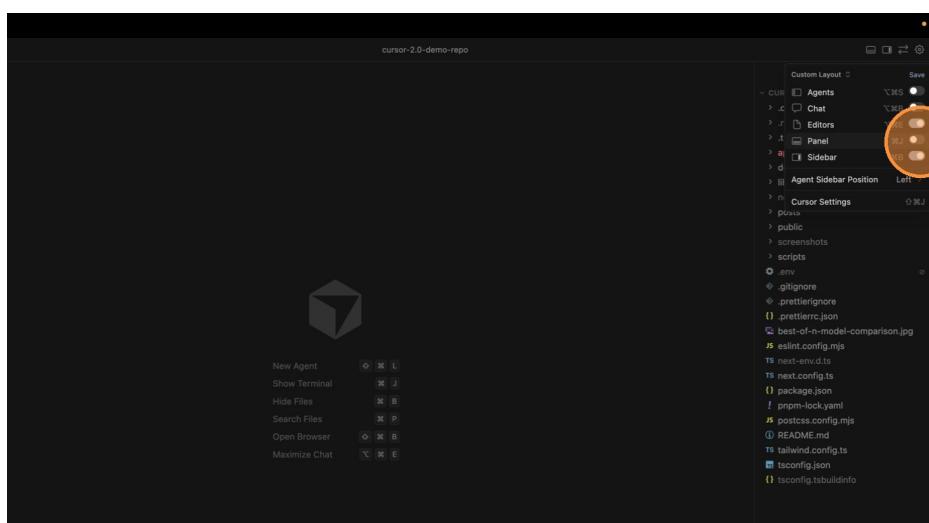
Layout options

Click the layout menu to see available options:



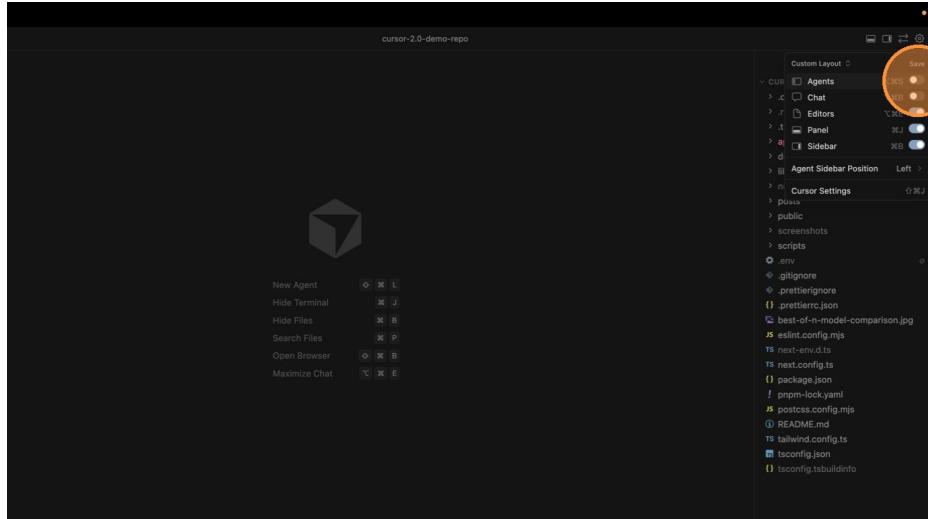
Layout menu

Click the File Explorer icon to see your project's file system:



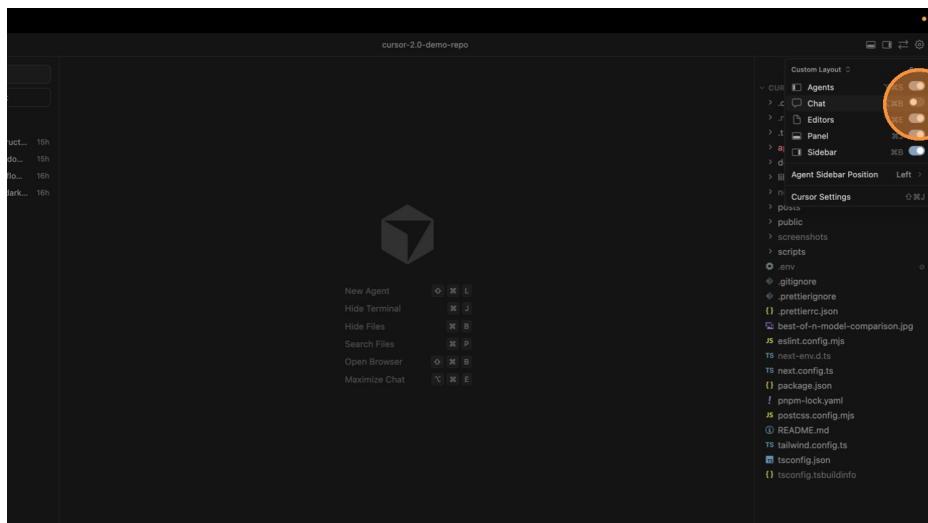
File Explorer panel

Click the Terminal area to open the integrated terminal:



Terminal panel

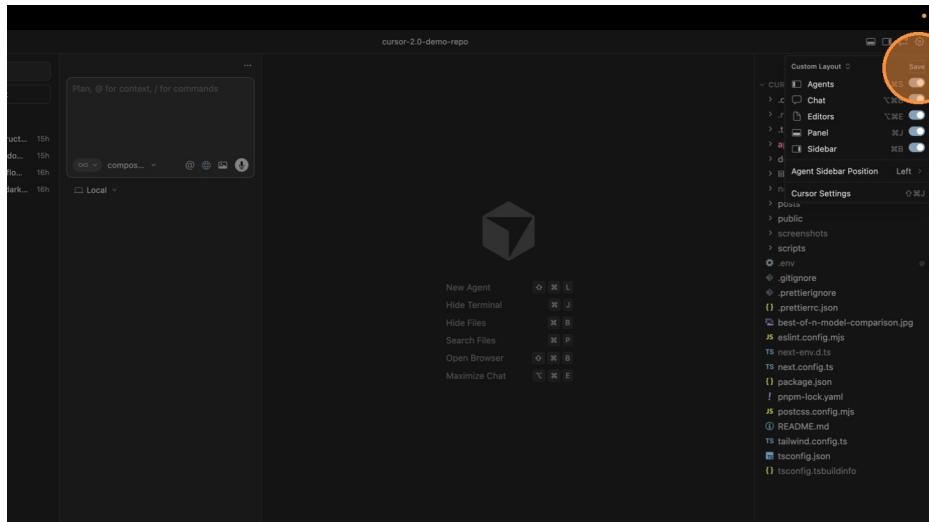
Click to open the Agent panel for AI chat and interactions:



Agent panel

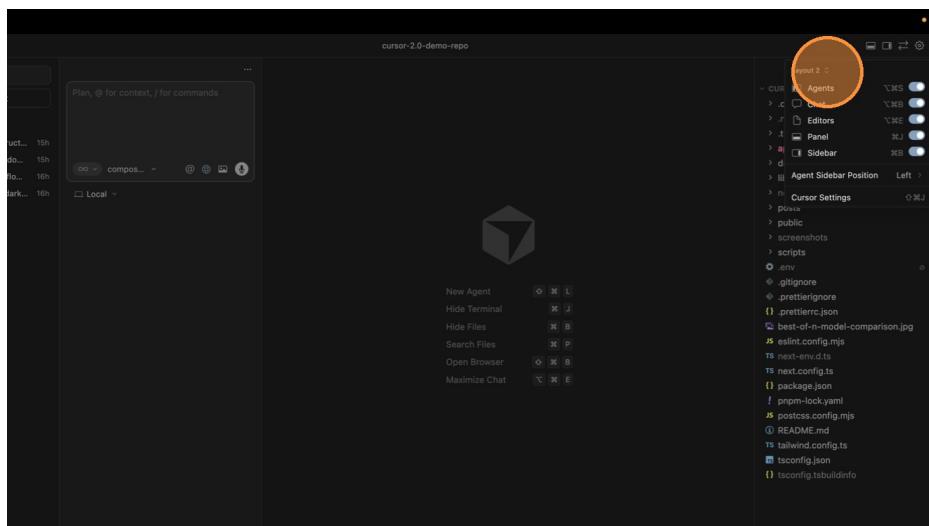
Saving Custom Layouts

Save your preferred layout for quick switching:



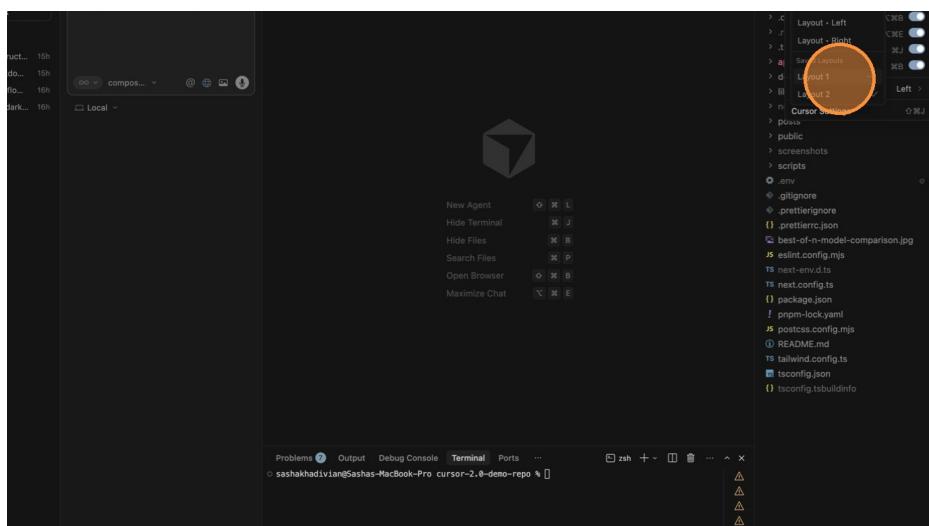
Save custom layout

Select a saved layout from the menu:



Select saved layout

Switch between different layouts as needed:



Switch layouts

What to Highlight

- Familiar VS Code experience with AI enhancements
- Customizable panel arrangement
- Quick layout switching for different workflows
- Integrated terminal for running commands alongside AI

Tips

- Use keyboard shortcuts to toggle panels quickly
 - Save different layouts for coding vs. AI-assisted tasks
 - The agent panel can be resized or detached
-

2. Tab + Quick Edit (Cmd+K)

Cursor Tab provides context-aware, multi-line code suggestions as you type. It can modify multiple lines at once, add import statements when missing, and predict your next editing location within or across files.

Inline Edit (Cmd+K) lets you edit code or generate new code directly within your editor using natural language. With code selected, it edits that specific code; without a selection, it generates new code at your cursor position.

Docs: [Tab Overview](#) | [Inline Edit](#)

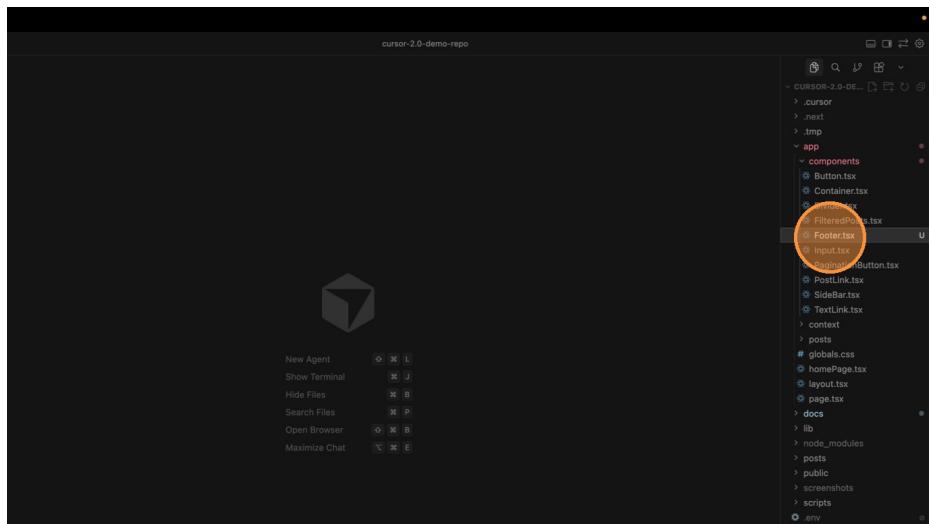
Tab Autocomplete

Overview

Tab predicts what you want to write next and suggests complete code blocks. The more you use it, the more it learns your style.

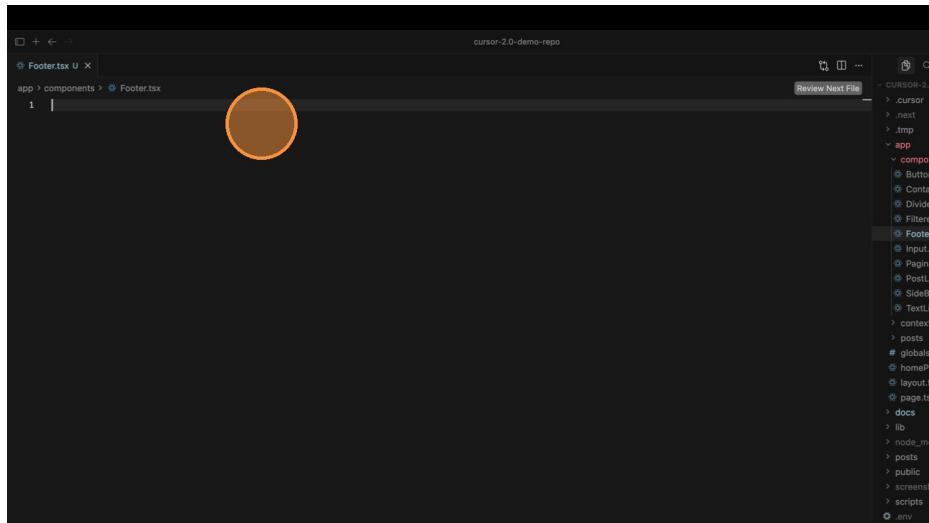
Demo

Open an empty component file like `Footer.tsx`:



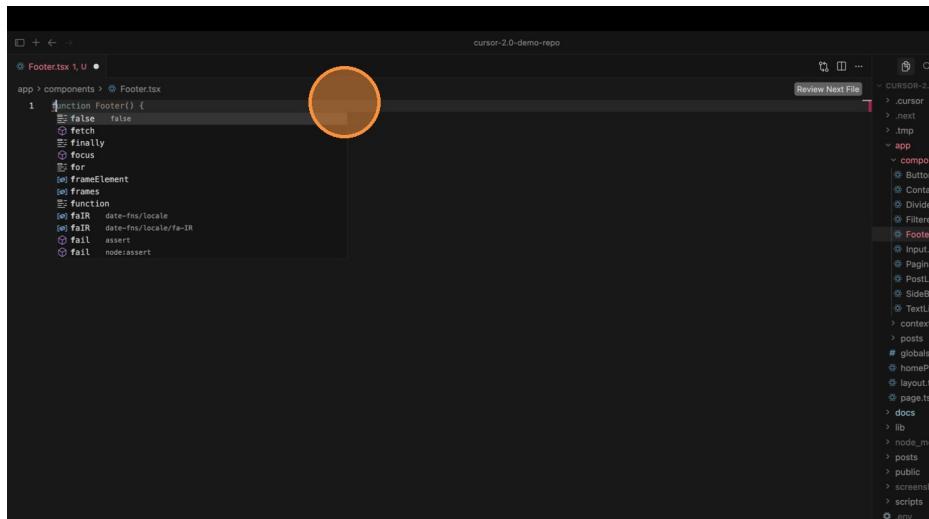
Open Footer.tsx

Type the letter “F”—Tab detects you’re in **Footer.tsx** and suggests the component:



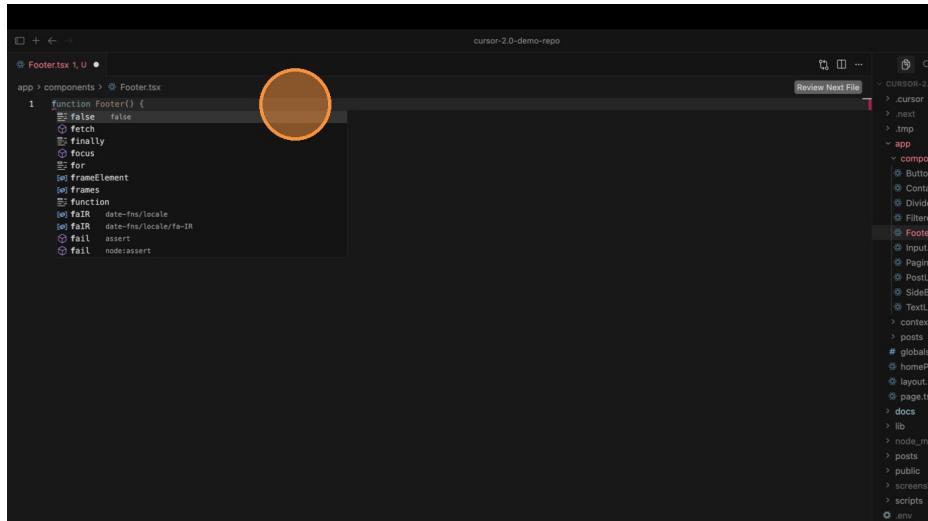
Start typing F

Tab suggests a complete Footer component structure:



Tab suggestion appears

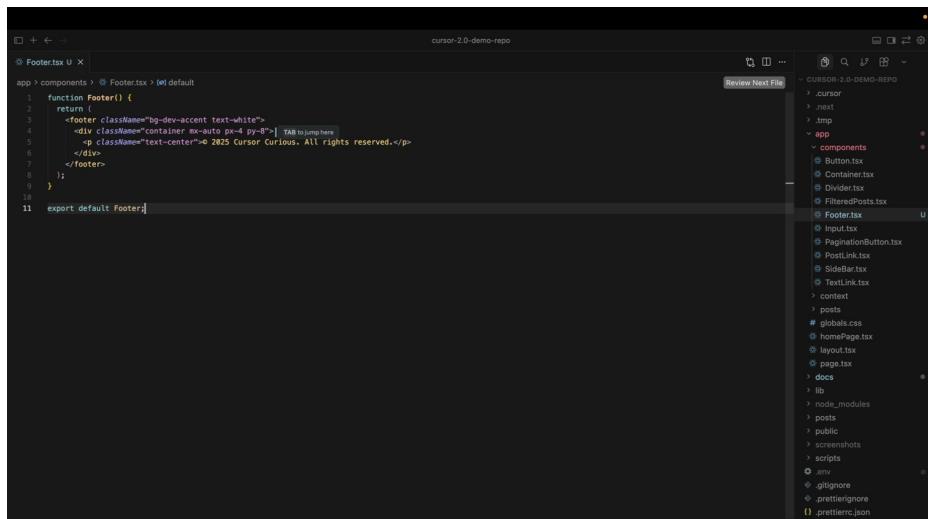
Press Tab to accept and continue building:



```
cursor-2.0-demo-repo
File Footer.tsx 1, U
app > components > Footer.tsx
1 function Footer() {
  false
  > fetch
  > finally
  > focus
  > for
  > frameElement
  > frames
  > function
  > faIR date-fns/locale
  > faIR date-fns/locale/fa-IR
  > fail assert
  > fail node:assert
```

Accept Tab suggestion

Keep pressing Tab—it builds out the component iteratively:



```
cursor-2.0-demo-repo
File Footer.tsx 1, U
app > components > Footer.tsx > default
1 function Footer() {
2   return (
3     <footer className="bg-dev-accent text-white">
4       <div className="container mx-auto px-4 py-8">| TAB to jump here
5         <p> className="text-center">&copy; 2025 Cursor Curious. All rights reserved.</p>
6       </div>
7     </footer>
8   );
9 }
10
11 export default Footer;
```

Continue with Tab

Click on the line above the closing footer tag and type “sign up”—Tab starts to auto-complete:

```

app > components > Footer.tsx > Footer
1 import Container from "./Container";
2
3 function Footer() {
4   return (
5     <Footer className="bg-dev-accent text-white">
6       <Container>
7         <p> className="text-center">© 2025 Cursor Curious. All rights reserved.</p>
8       </Container>
9       Sign up for our newsletter
10    </Footer>
11  );
12}
13 export default Footer;
14
15 // Scripts
16 // Selectors
17 // Selectors
18 // Selectors
19 // Small
20 // Source

```

Type sign up hint

Press Tab to accept—it uses components from your codebase to build the feature:

```

> components > Footer.tsx > Footer
import Button from "./Button";
import Container from "./Container";
import Input from "./Input";

function Footer() {
  return (
    <Footer className="bg-dev-accent text-white">
      <Container>
        <p> className="text-center">© 2025 Cursor Curious. All rights reserved.</p>
      </Container>
      Sign up for our newsletter
      <Input
        value=""
        onChange={() => {}}
        onInput={() => {}}
        placeholder="Enter your email"
      />
      <Button onClick={() => {}} variant="primary" className="ml-2 w-full">Sign up</Button>
    </Footer>
  );
}

export default Footer;

```

Tab uses existing components

Continue pressing Tab to iterate quickly—express ideas and Tab builds them out:

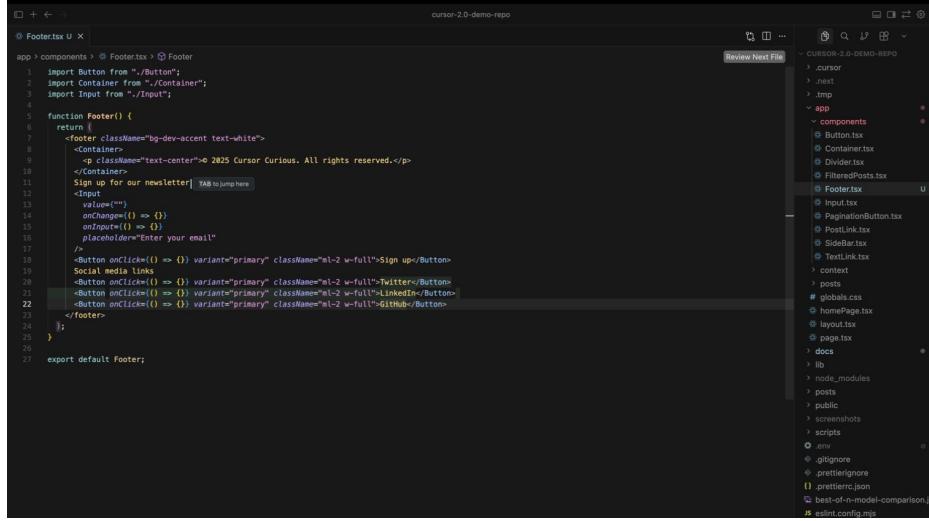
```

@ Footer.tsx U
app > components > Footer.tsx > Footer
1 import Button from "./Button";
2 import Container from "./Container";
3 import Input from "./Input";
4
5 function Footer() {
6   return (
7     <Footer className="bg-dev-accent text-white">
8       <Container>
9         <p> className="text-center">© 2025 Cursor Curious. All rights reserved.</p>
10        Sign up for our newsletter
11        <Input
12          value=""
13          onChange={() => {}}
14          onInput={() => {}}
15          placeholder="Enter your email"
16        />
17        <Button onClick={() => {}} variant="primary" className="ml-2 w-full">Sign up</Button>
18      </Container>
19    );
20  }
21
22
23
24 export default Footer;

```

Rapid iteration with Tab

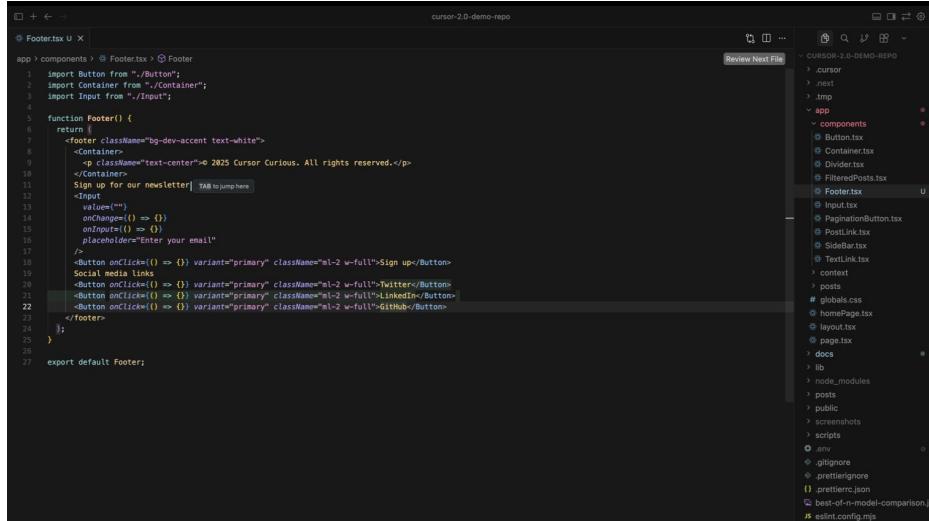
Type “social media links” and Tab adapts to your intent—press Tab as it auto-completes:



```
cursor-2.0-demo-repo
Review Next File
cursor-2.0-demo-repo
Footer.tsx
app > components > Footer.tsx > Footer
1 import Button from './Button';
2 import Container from './Container';
3 import Input from './Input';
4
5 function Footer() {
6   return (
7     <footer className="bg-dev-accent text-white">
8       <Container>
9         <p>2025 Cursor Curious. All rights reserved.</p>
10        <div>
11          Sign up for our newsletter! TAB to jump here
12          <Input
13            value=""
14            onChange={() => {}}
15            onInput={() => {}}
16            placeholder="Enter your email"
17            />
18          <Button onClick={() => {}} variant="primary" className="ml-2 w-full">Sign up</Button>
19        </div>
20        <SocialMediaLinks>
21          <SocialMediaLink></SocialMediaLink>
22          <SocialMediaLink></SocialMediaLink>
23          <SocialMediaLink></SocialMediaLink>
24          <SocialMediaLink></SocialMediaLink>
25        </SocialMediaLinks>
26      </footer>
27    )
28  }
29
30  export default Footer;
```

Type social media links

Tab now uses social links instead of the signup form:



```
cursor-2.0-demo-repo
Review Next File
cursor-2.0-demo-repo
Footer.tsx U
app > components > Footer.tsx > Footer
1 import Button from './Button';
2 import Container from './Container';
3 import Input from './Input';
4
5 function Footer() {
6   return (
7     <footer className="bg-dev-accent text-white">
8       <Container>
9         <p>2025 Cursor Curious. All rights reserved.</p>
10        <div>
11          Sign up for our newsletter! TAB to jump here
12          <Input
13            value=""
14            onChange={() => {}}
15            onInput={() => {}}
16            placeholder="Enter your email"
17            />
18          <Button onClick={() => {}} variant="primary" className="ml-2 w-full">Sign up</Button>
19        </div>
20        <SocialMediaLinks>
21          <SocialMediaLink></SocialMediaLink>
22          <SocialMediaLink></SocialMediaLink>
23          <SocialMediaLink></SocialMediaLink>
24          <SocialMediaLink></SocialMediaLink>
25        </SocialMediaLinks>
26      </footer>
27    )
28  }
29
30  export default Footer;
```

Tab adapts to changes

What to Highlight

- Contextual awareness—knows what file you’re in
- Uses your existing components and patterns
- Gets smarter the more you use it
- Feels like “reading your mind” for power users

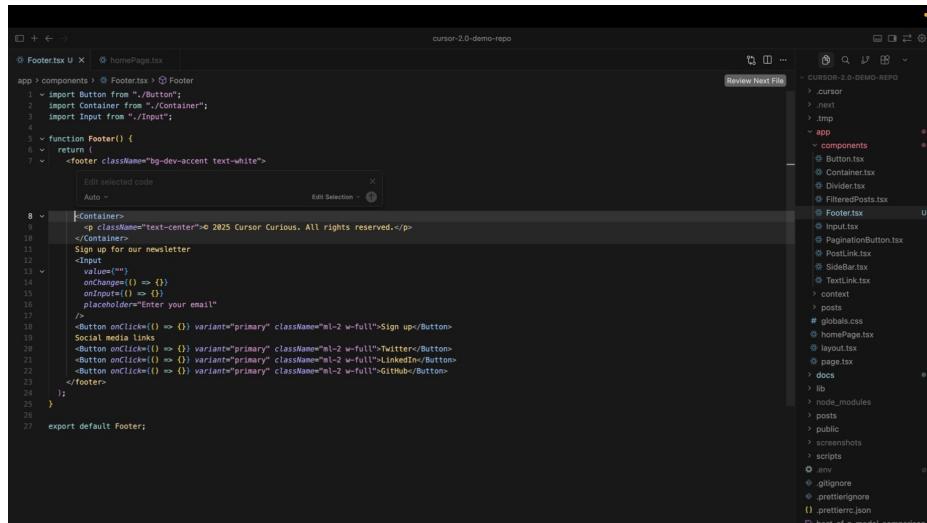
Quick Edit (Cmd+K)

Overview

Select code and press Cmd+K to make targeted edits using natural language. Only the selected section is modified.

Demo

Select a section of code you want to edit and press Cmd+K to open the quick edit panel:

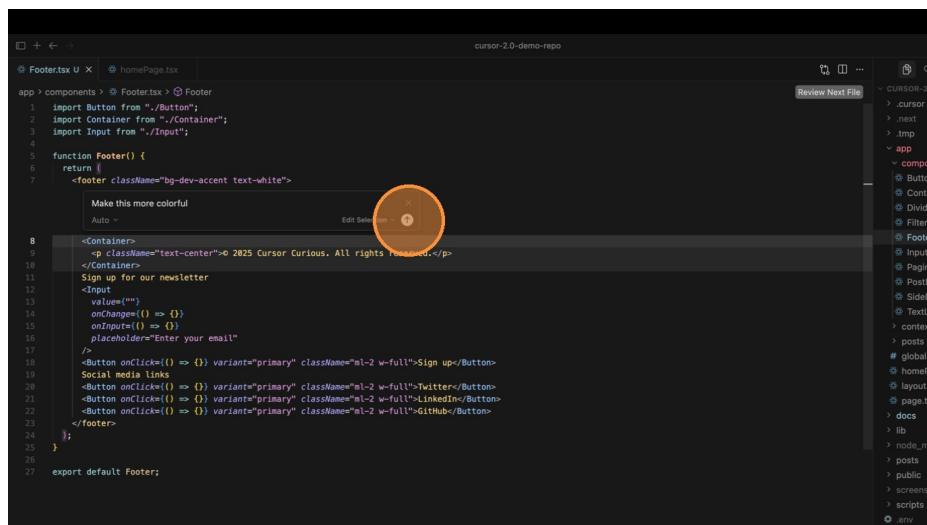


The screenshot shows a code editor with a file named 'Footer.tsx' open. A specific section of the code is selected, and a small 'Edit Selection' button is visible. A tooltip labeled 'Edit Selected code' appears above the button. The code block is as follows:

```
1 <function Footer() {
2   return (
3     <footer className="bg-dev-accent text-white">
4       <Container>
5         <p>2025 Cursor Curious. All rights reserved.</p>
6         <Sign up for our newsletter>
7           <Input
8             value=""
9             onChange={() => {}}
10            onInput={() => {}}
11            placeholder="Enter your email"
12           />
13           <Button onClick={() => {}} variant="primary" className="ml-2 w-full">Sign up</Button>
14           Social media links
15           <Button onClick={() => {}} variant="primary" className="ml-2 w-full">Twitter</Button>
16           <Button onClick={() => {}} variant="primary" className="ml-2 w-full">LinkedIn</Button>
17           <Button onClick={() => {}} variant="primary" className="ml-2 w-full">GitHub</Button>
18         </Container>
19       </footer>
20     )
21   }
22   export default Footer;
23 }
```

Quick edit prompt

Type your instruction—“make this more colorful”:



The screenshot shows the same code editor interface, but now the 'Edit Selection' button is highlighted with a red circle. The tooltip 'Edit Selected code' is still present. The code block remains the same as in the previous screenshot.

Type instruction

Cursor edits only the selected section and shows the diff:

```

1 import Button from "./Button";
2 import Container from "./Container";
3 import Input from "./Input";
4
5 function Footer() {
6   return (
7     <footer className="bg-dev-accent text-white">
8       <Container>
9         <p>2025 Cursor Curious. All rights reserved.</p>
10        <p>Cursor accent color</p>
11        <Text>
12          color: "#f54e80"; // Cursor accent color
13          textShadow: "0 2px 6px rgba(30,37,38,0.18)", // subtle shadow using primary fg
14          background: "linear-gradient(90deg, #f54e80 0%, #26251e 100%)",
15          WebkitBackgroundClip: "text",
16          WebkitTextFillColor: "transparent",
17          display: "inline-block",
18        </Text>
19      </Container>
20      <Text>
21        <span style={{ color: "#26251e" }}>Cursor</span> All rights reserved.
22      </Text>
23    </Container>
24    Sign up for our newsletter
25    <Input
26      value=""
27      onChange={() => {}}
28      onInput={() => {}}
29      placeholder="Email address"
30    />

```

[View the diff](#)

What to Highlight

- Surgical edits—only changes what you select
- Natural language instructions
- Preview changes before accepting
- Great for refactoring specific sections

Best Practices

- **Tab:** Start with hints in comments or partial code to guide suggestions
- **Quick Edit:** Be specific about what you want changed
- **Combine them:** Use Tab for building, Quick Edit for refining

3. Agent Chat

Agent Chat is the command center for agentic workflows within Cursor. An agent is a large language model (LLM) with access to tool calls—in this case, the ability to make changes directly to your codebase.

[Docs: Agent Chat](#)

Overview

This demo covers all the key configurations and features of Agent Chat.

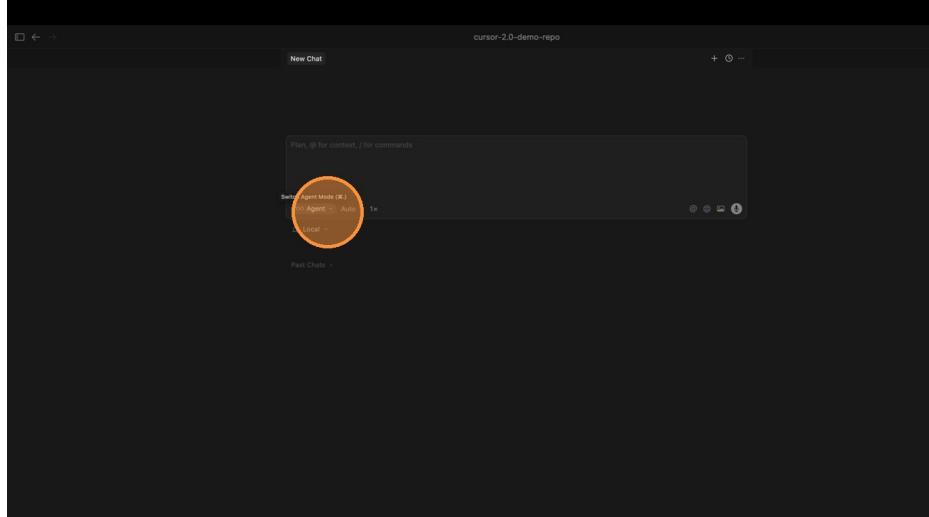
3.1 Agent Modes

Cursor provides different modes optimized for different tasks. Choose the right mode for your workflow.

Mode	Use Case
Agent	

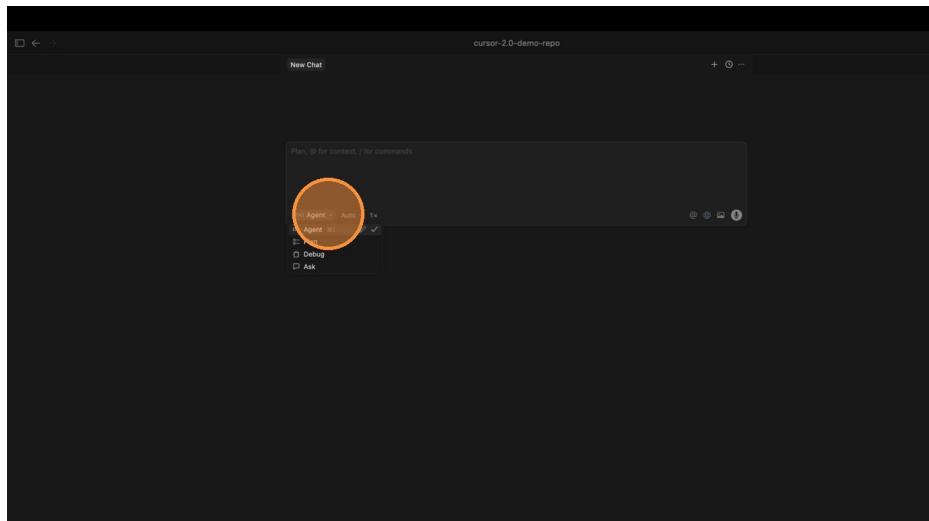
Mode	Use Case
	Execute code changes across your codebase (default)
Plan	Create editable implementation plans for complex features
Debug	Find, test, and resolve difficult bugs
Ask	Learn about the codebase without making changes (read-only)

Click the mode dropdown to see available options:



Agent mode selector

Select from Agent, Plan, Debug, or Ask mode:



Mode options

3.2 Model Selection

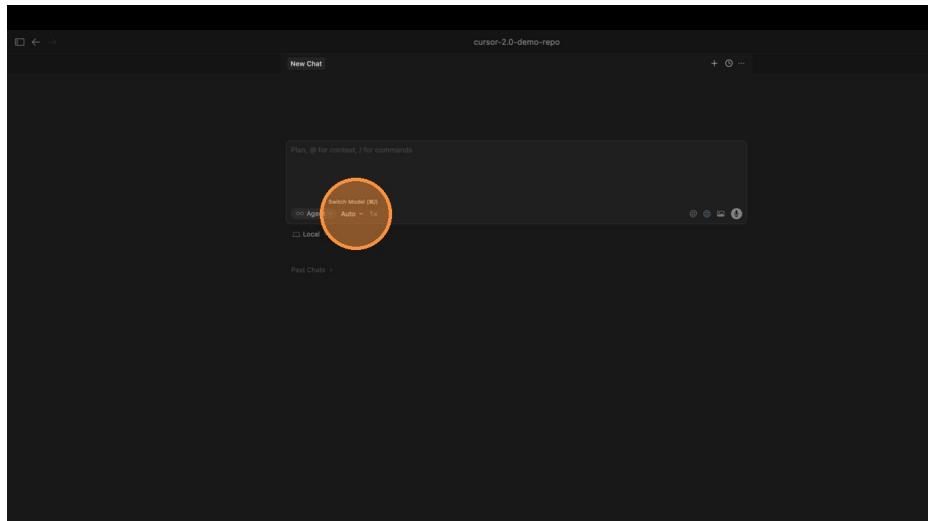
You have full control over which model to use. Choose based on your task complexity and speed requirements.

- **Composer 1:** Cursor's own coding model, optimized for speed

- **Auto:** Let Cursor smartly select the best model for each task
- **Frontier models:** Claude, GPT-4, Gemini, and more

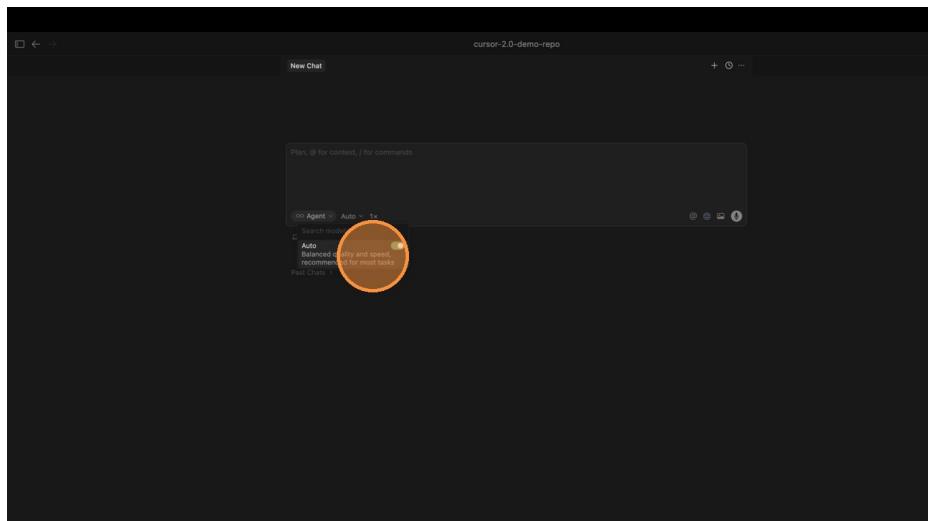
Quick Selection

Click the model selector:



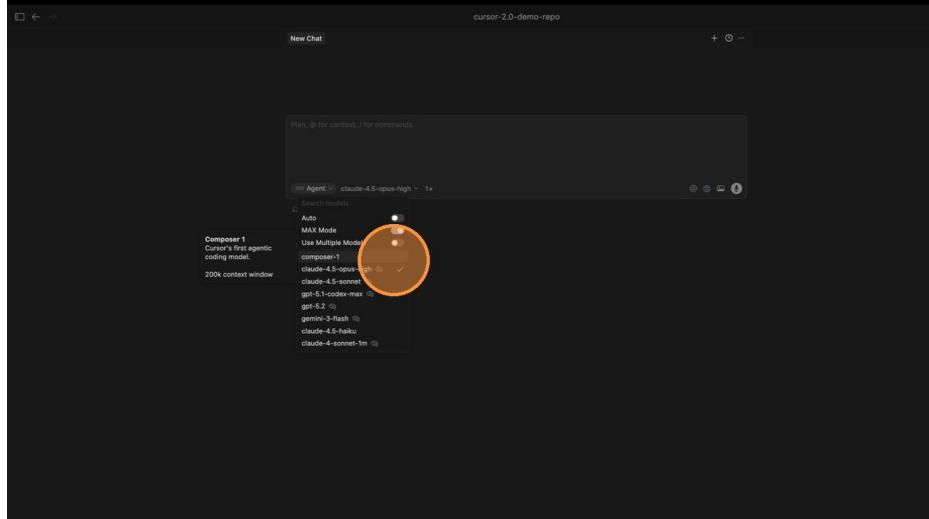
Model selector

Choose from available models:



Model options with descriptions

Model categories and Auto selection:



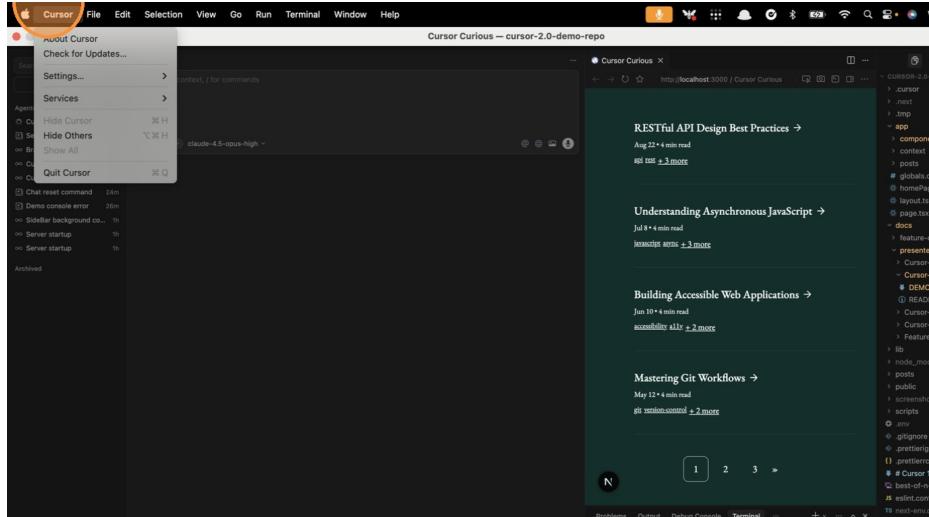
Model categories

Model Settings

For more control, configure models in **Cursor Settings > Models**:

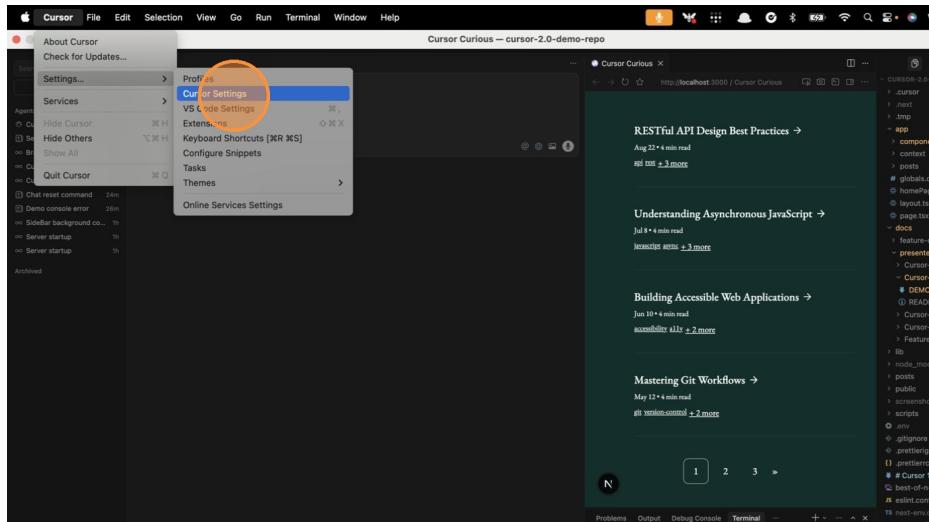
1. Open **Cursor** menu > **Cursor Settings**
2. Navigate to **Models**
3. Click **View All Models** to see all available options
4. Enable or disable models based on your needs

Open Cursor Settings:



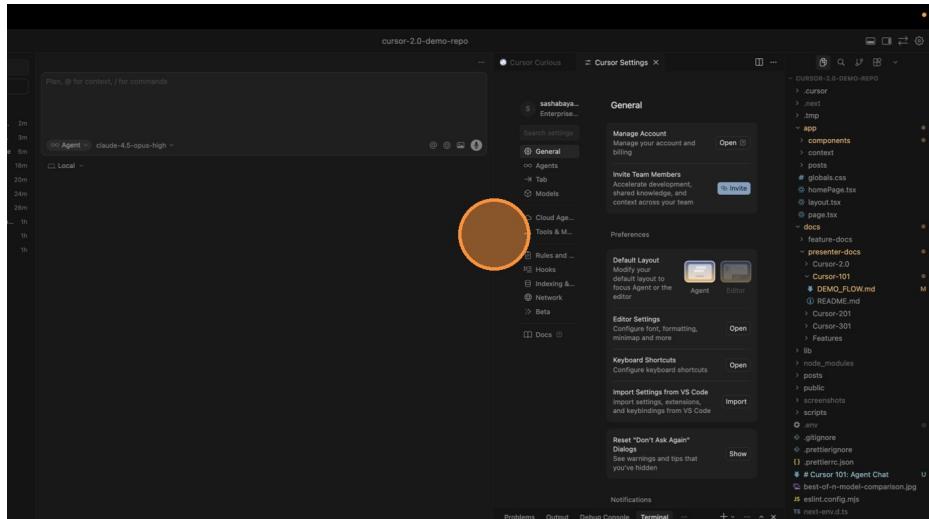
Cursor menu

Click Cursor Settings:



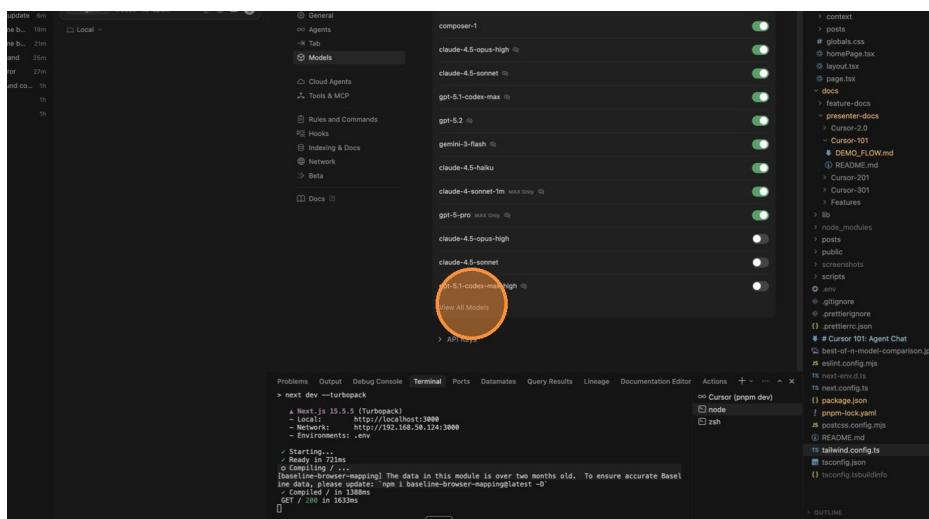
Cursor Settings option

Navigate to Models section:



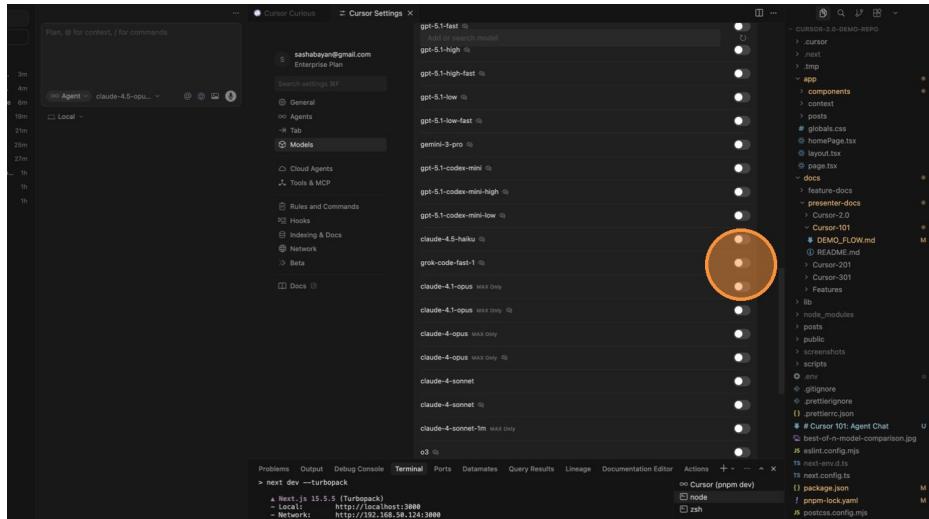
Models section

Click View All Models:



View All Models button

Enable or disable models:



Model list with toggles

3.3 Adding Context

| Provide the agent with relevant context so it can make informed decisions.

Why This Matters

Context is everything when working with AI. Without the right context, even the best model will give generic or incorrect answers. The friction of adding context—switching between apps, copying documentation from ChatGPT or Claude’s web interface, pasting error messages—creates constant context switching that slows you down more than you realize.

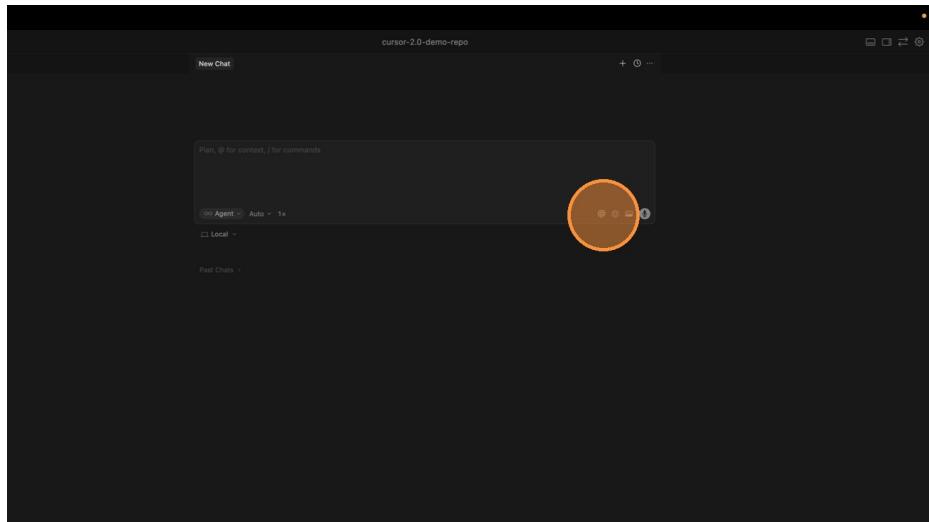
Cursor makes adding context fast and seamless. Instead of juggling browser tabs and copy-pasting between applications, you simply type @ and reference what you need. This eliminates the hidden productivity tax of context switching, letting you stay in flow while giving the agent exactly what it needs to help you.

How to Add Context

Use the **Add Context** button (or type @) to include:

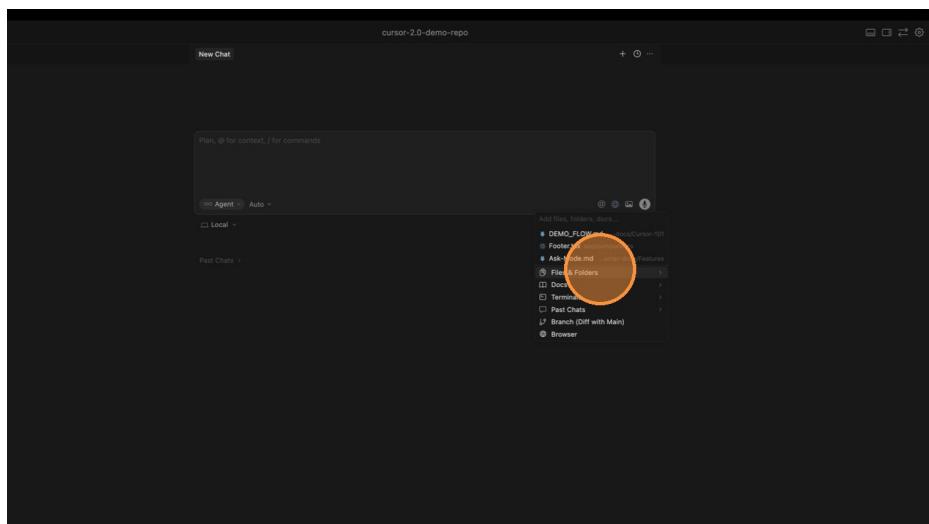
- **Files** – Specific files to reference or modify
- **Folders** – Entire directories for broader context
- **Docs** – Up-to-date documentation from popular libraries
- **Terminal** – Terminal output for debugging errors
- **Past Chats** – Reference previous conversations

Click the Add Context button:



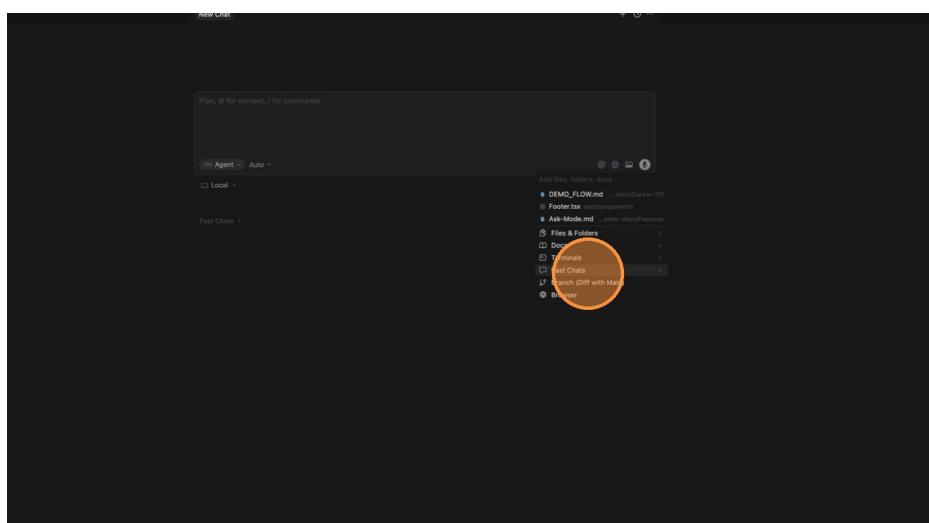
Add Context button

Context options menu:



Context options

Available context types:



Context types expanded

3.4 Image Upload

Upload images as reference for the agent—mockups, screenshots, or error messages.

Click the image upload button:

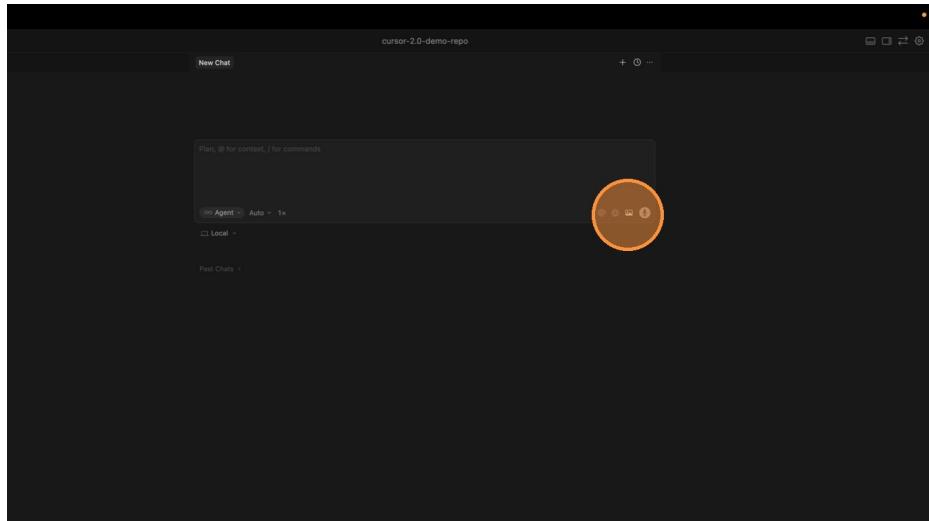


Image upload button

Image attached to chat:

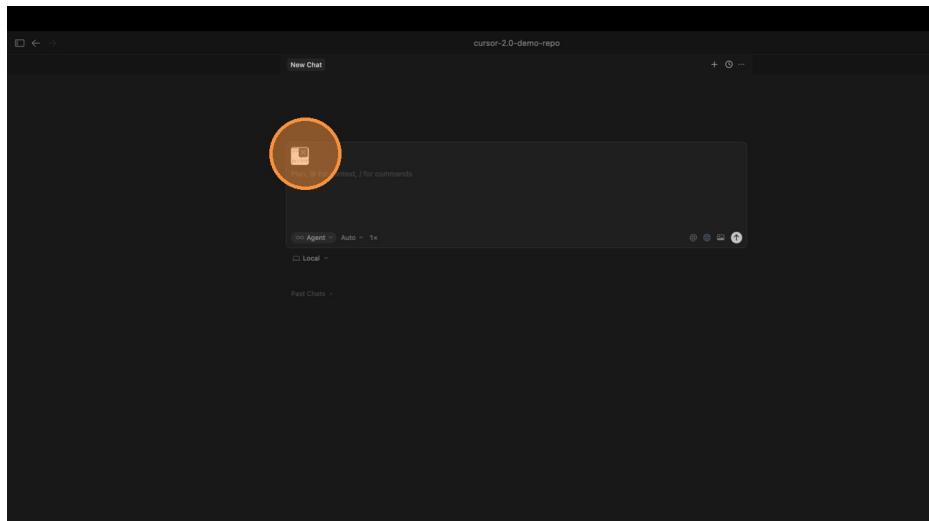
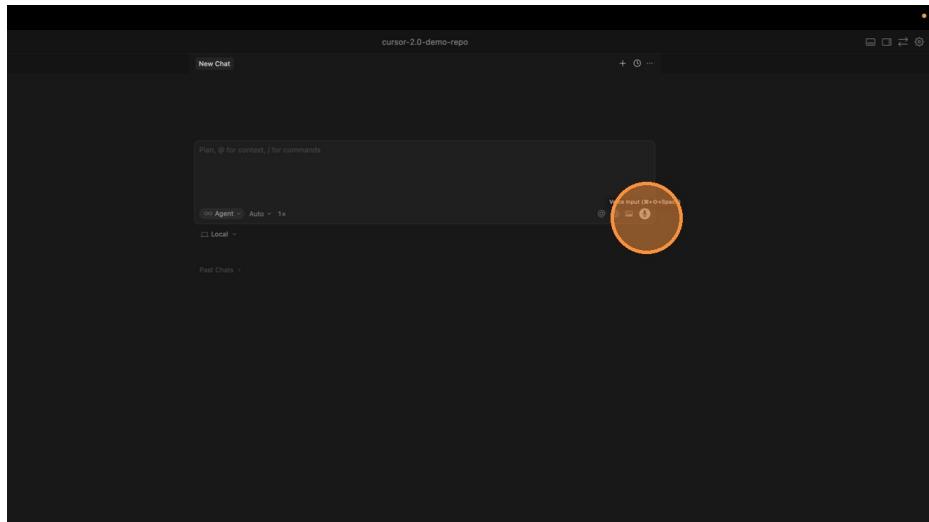


Image in chat

3.5 Voice Input

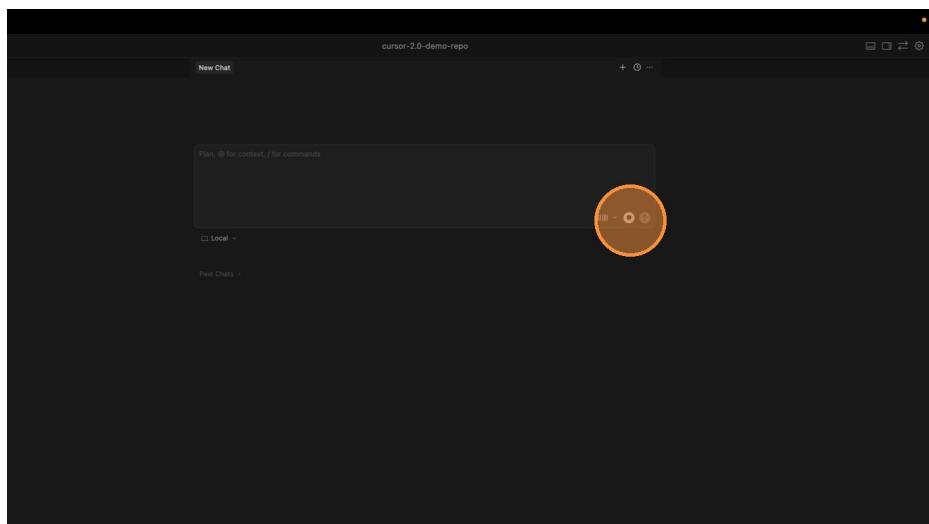
Speak your prompts instead of typing for faster iteration.

Click the microphone button to record:



Voice input button

Recording in progress:



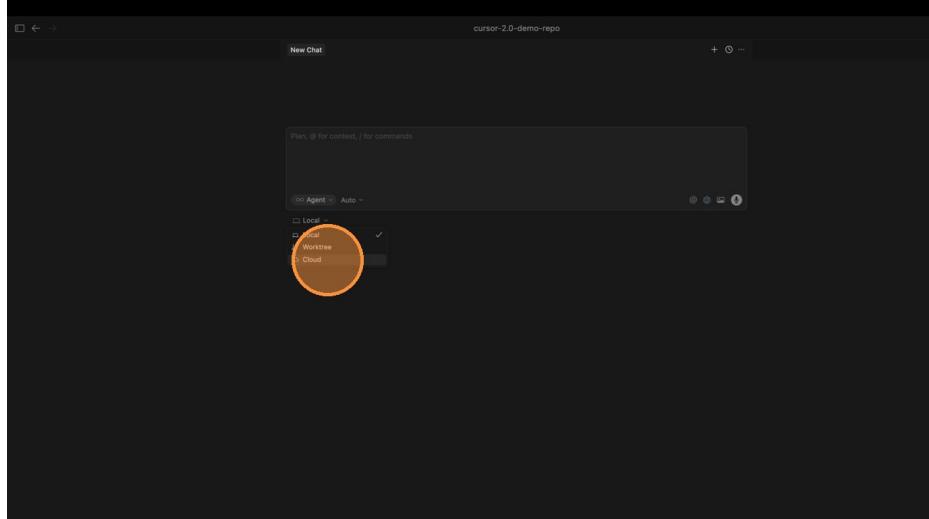
Voice recording active

3.6 Execution Mode

Choose where your agent runs based on your needs.

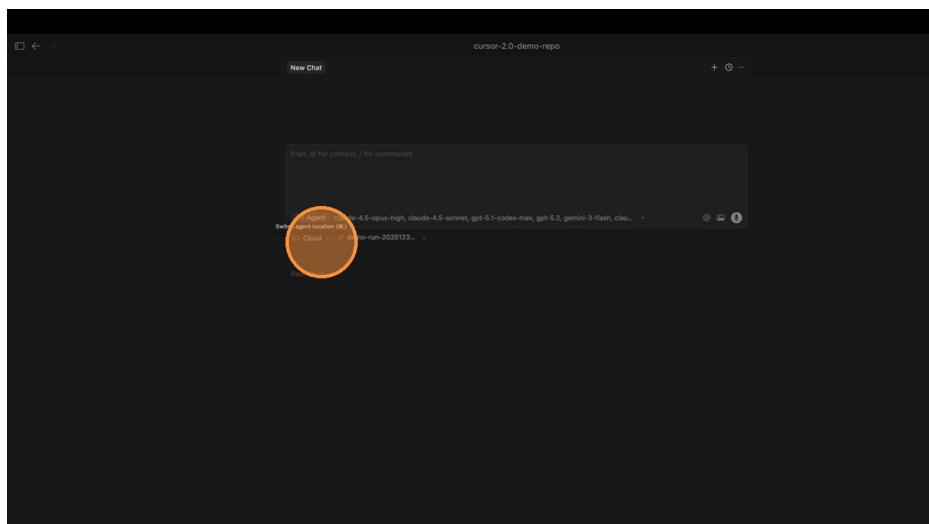
Mode	Description
Local	Runs on your machine with full access to local files
Cloud	Runs on Cursor's cloud infrastructure
Worktree	Run parallel agents in isolated worktrees (advanced)

Local execution mode:



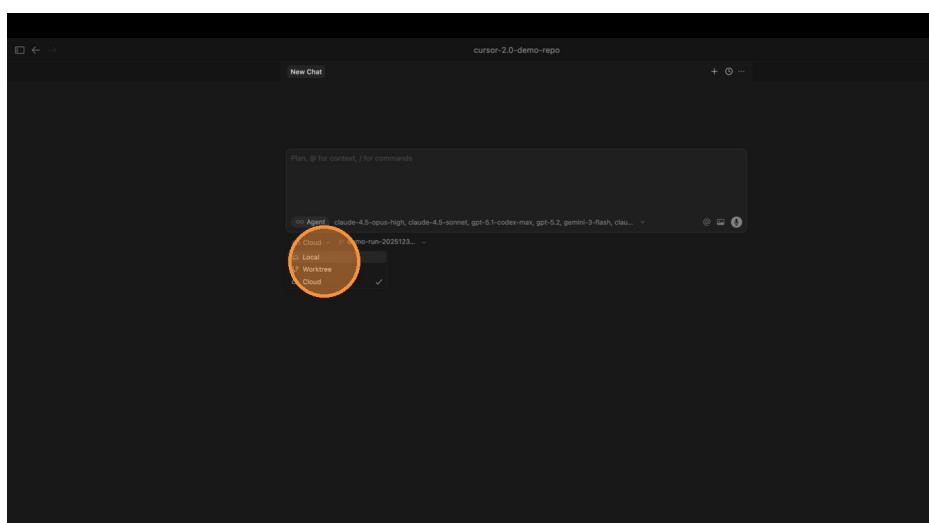
Local mode

Cloud execution mode:



Cloud mode

Worktree mode (advanced):



Worktree mode

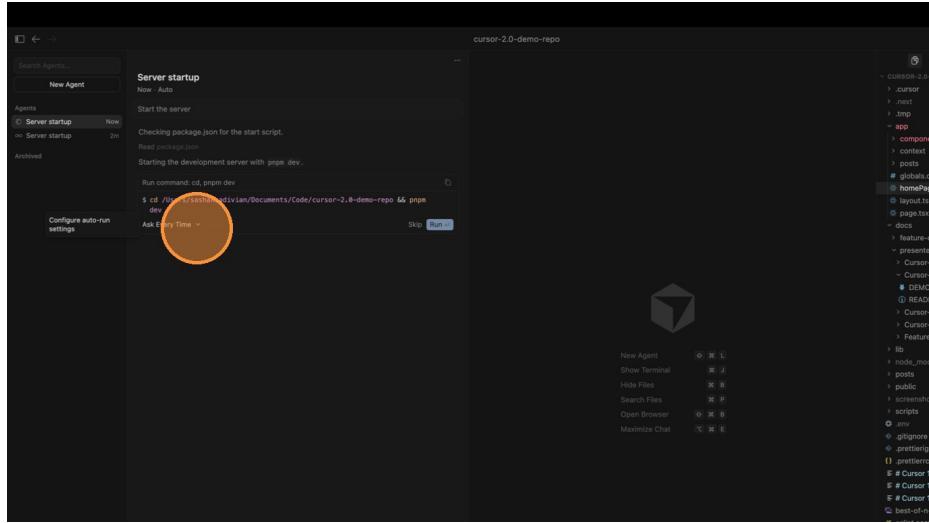
3.7 Starting the Server

Let the agent handle common tasks like starting your development server.

- Start the development server

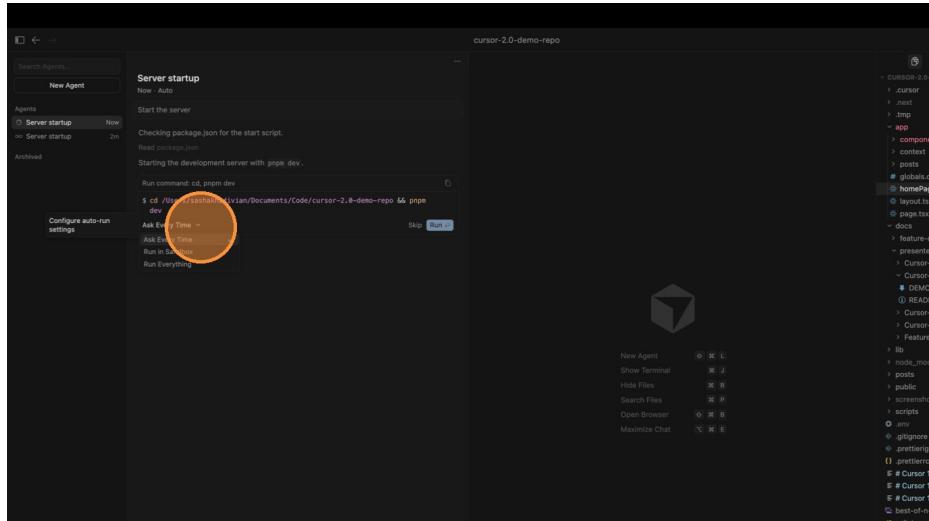
The agent will inspect your project configuration and run the appropriate command.

Agent requests permission to run terminal command:



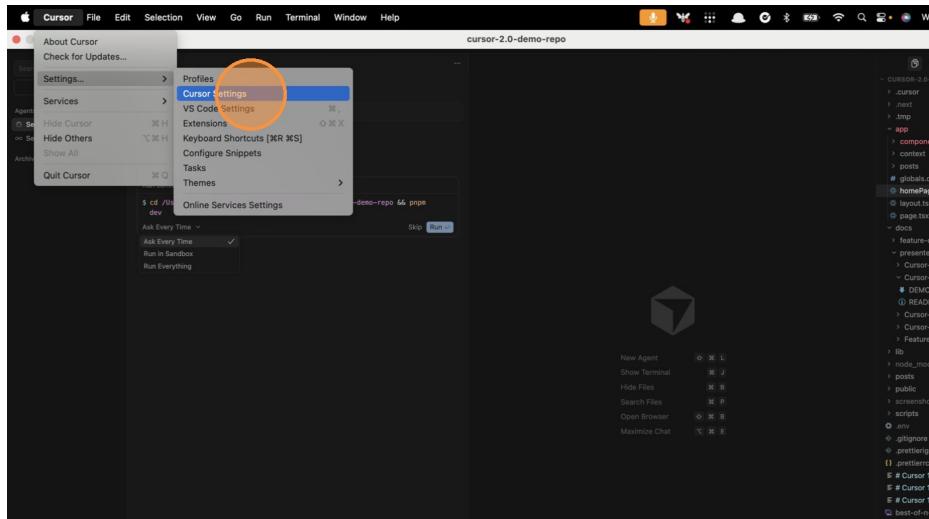
Terminal permission request

Terminal command options:

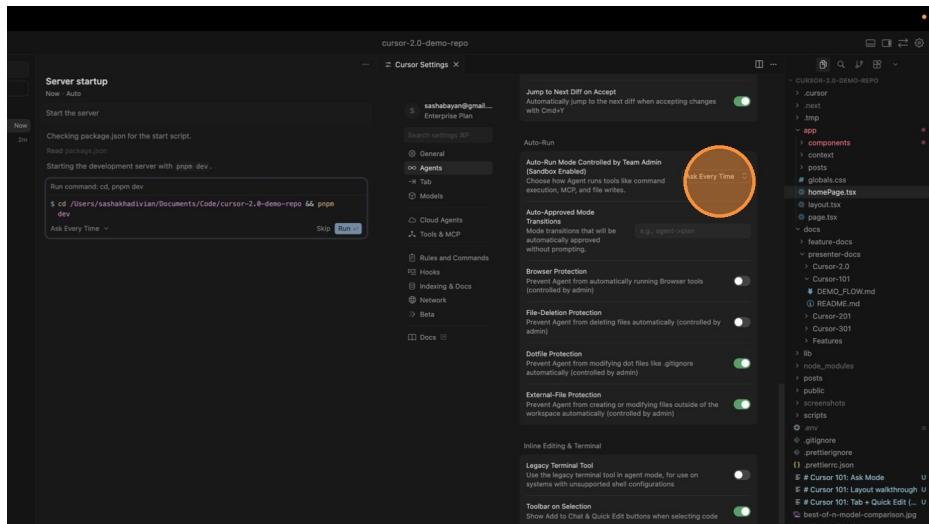


Terminal options - Ask, Sandbox, Run Everything

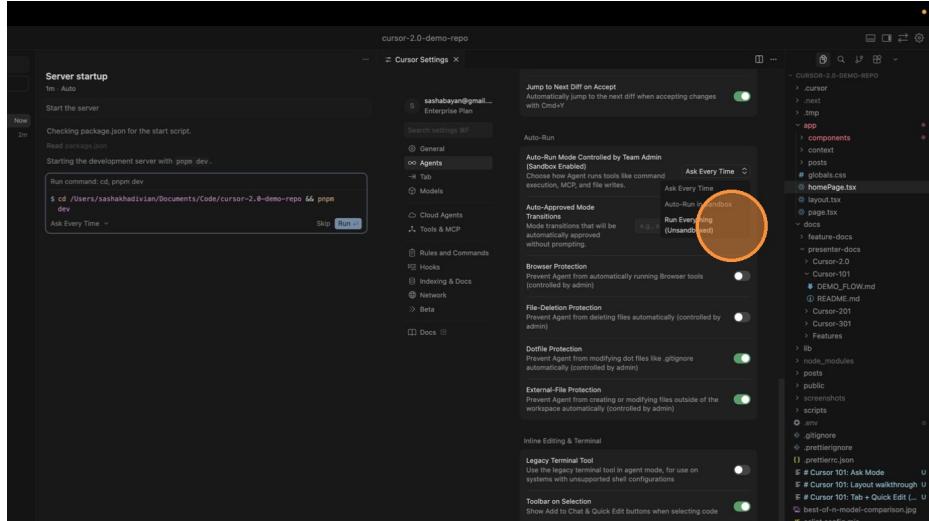
Configure default terminal permissions in Cursor Settings > Agents:



Cursor Settings menu

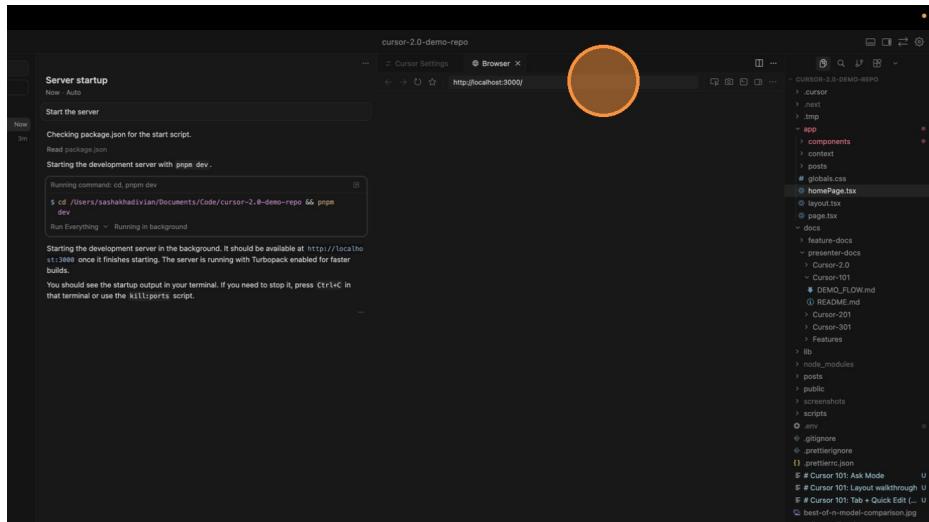


Agent settings



Terminal permission settings

Server running in browser:



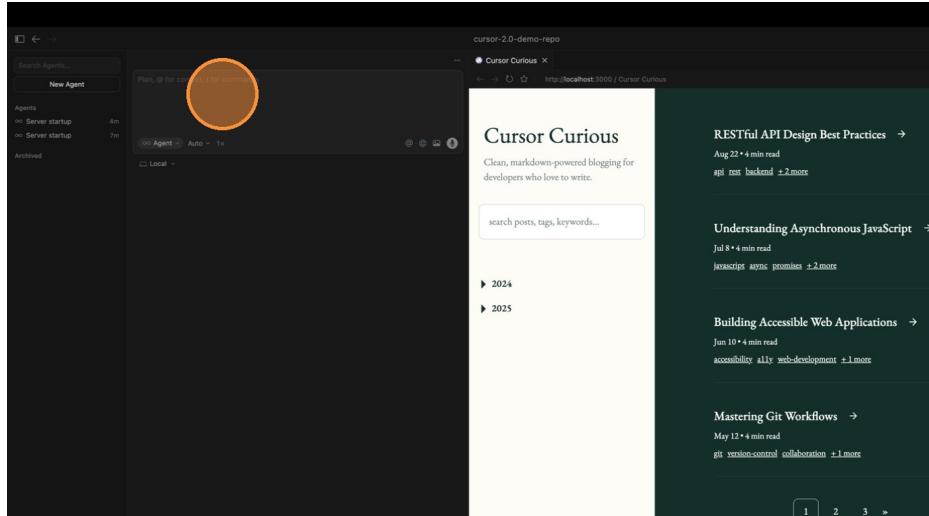
App running in browser

3.8 Making Code Changes

Reference specific files when asking for changes to help the agent find the right code immediately.

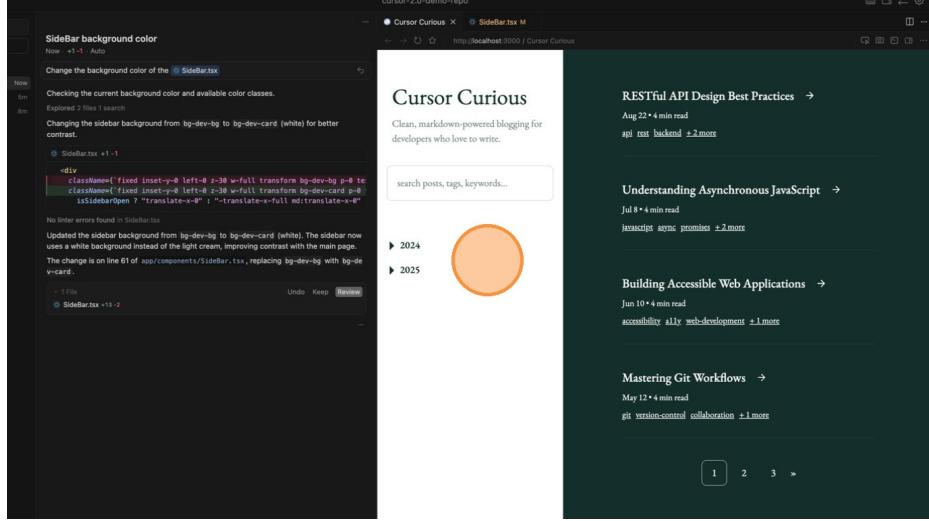
- Change sidebar background to blue

Reference a file in your prompt:



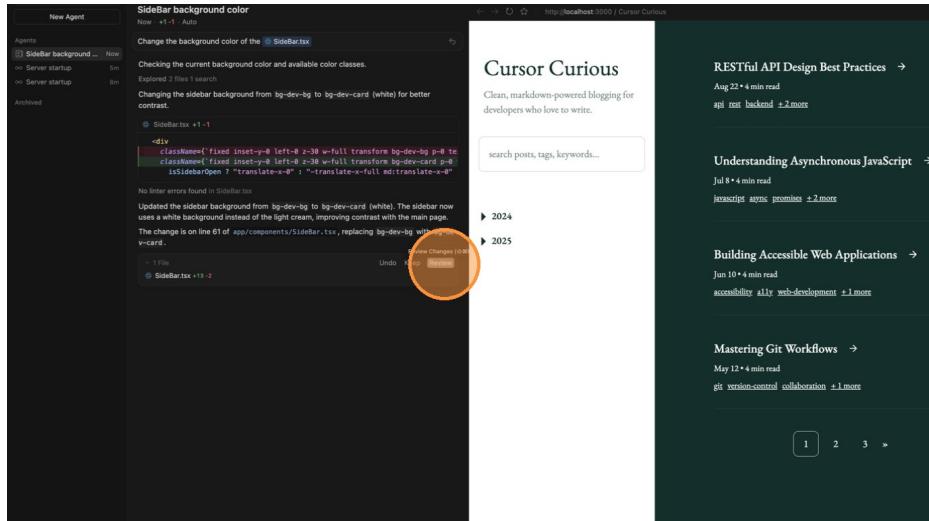
File reference in prompt

Agent identifies the correct component:



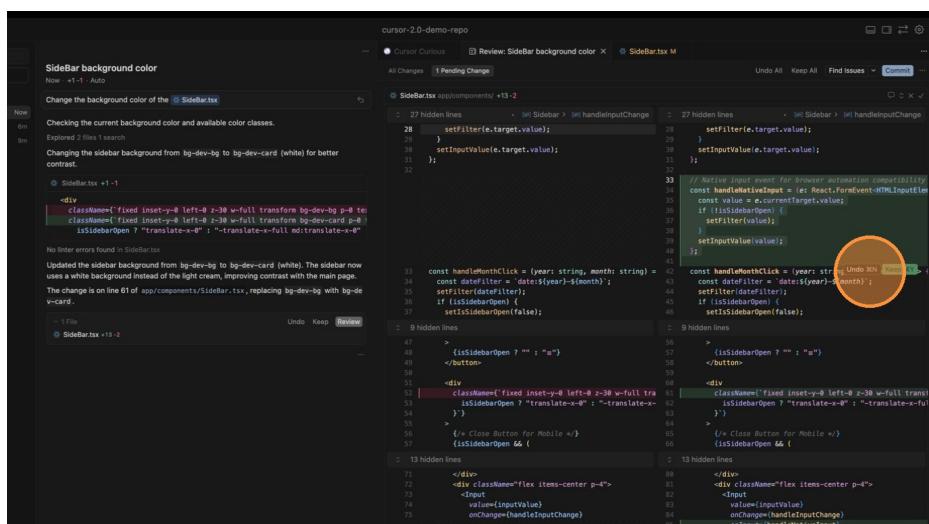
Agent finds component

Review changes with the diff view:



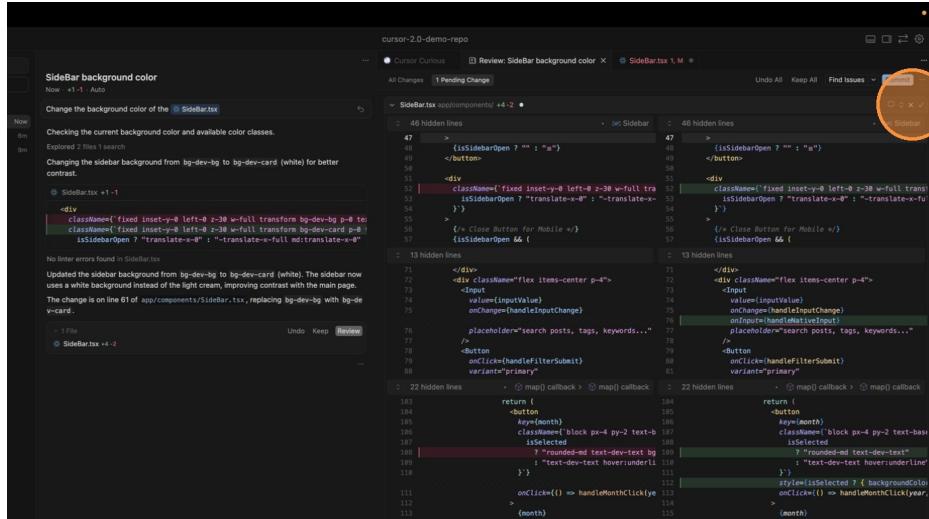
Review changes button

Diff view showing changes:



Diff view

Undo changes if needed:



The screenshot shows a code editor interface with a file named 'SideBar background color'. The code is related to sidebar styling, specifically changing the background from light cream to white. A circled 'Undo' button is located in the top right corner of the editor window.

Undo changes

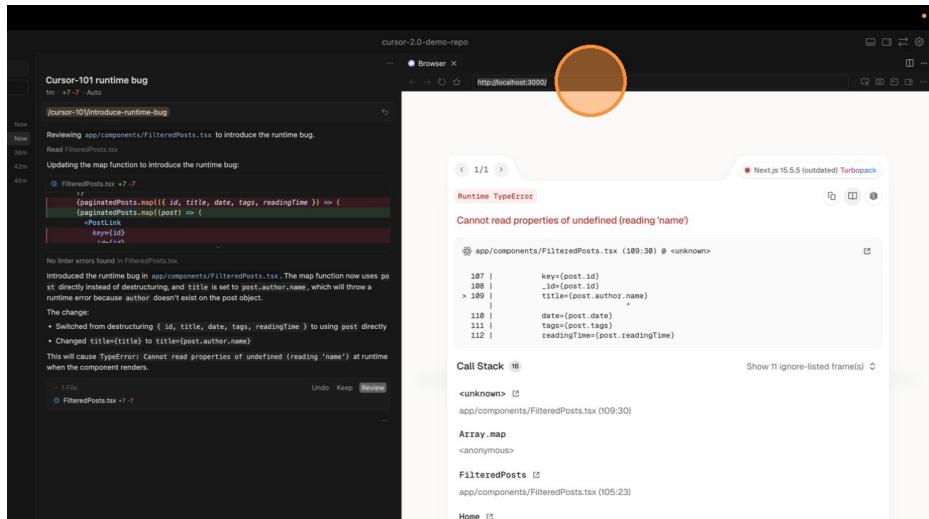
3.9 Fixing Bugs from Terminal Output

Quickly fix bugs by piping terminal errors directly into the agent.

Step 1: Introduce a bug (for demo purposes) - Use `/introduce-runtime-bug` slash command

Step 2: View the error

Error displayed in browser:



The screenshot shows a browser window with a runtime error message in the developer tools console. The error is a 'Runtime Type Error' stating 'Cannot read properties of undefined (reading 'name')'. The error occurs in the file 'app/components/FilteredPosts.tsx' at line 109, column 38. The console also shows a call stack and other relevant code snippets.

Runtime error in browser

Open the terminal panel:

The screenshot shows the Cursor IDE interface. At the top, there's a status bar with 'Cursor Settings' and a timestamp '0:36J'. Below it is a code editor window with a file named 'FilteredPosts.tsx'. The code contains a runtime error message: 'Done. The runtime bug has been introduced in FilteredPosts.tsx. The code now accesses post.author.name instead of post.title, which will cause a TypeError: Cannot read properties of undefined (reading 'name') when the component renders, since author doesn't exist on the post object.' The terminal tab is highlighted with a red circle.

Open terminal panel

Select the terminal with the error:

This screenshot shows the Cursor IDE interface with the terminal panel open. The terminal output shows a 'Runtime TypeError' with the message 'Cannot read properties of undefined (reading 'name')'. The cursor is positioned over the error message, and a red circle highlights the terminal output area.

Select terminal

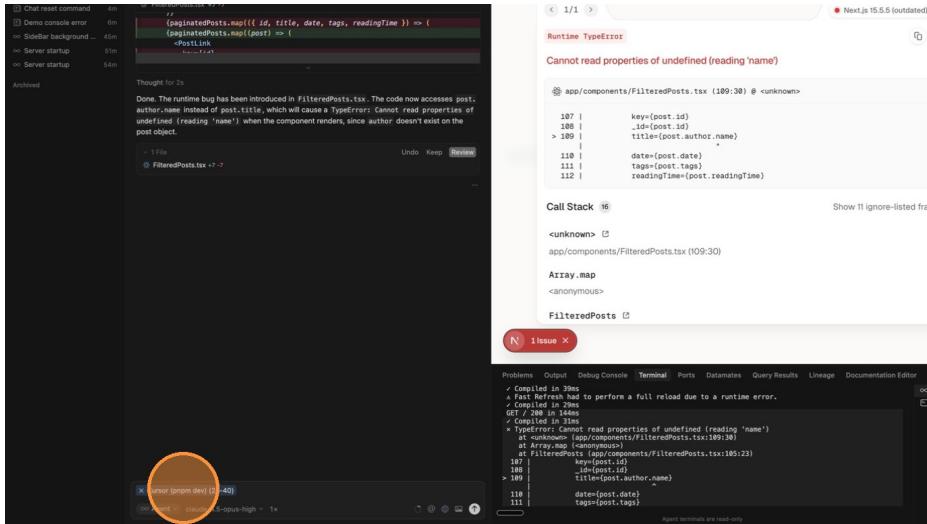
Step 3: Send error to agent

Select the error text:

This screenshot shows the Cursor IDE interface with the terminal panel open. The terminal output shows a 'Runtime TypeError' with the message 'Cannot read properties of undefined (reading 'name')'. The cursor is positioned over the error message, and a red circle highlights the terminal output area.

Select error text

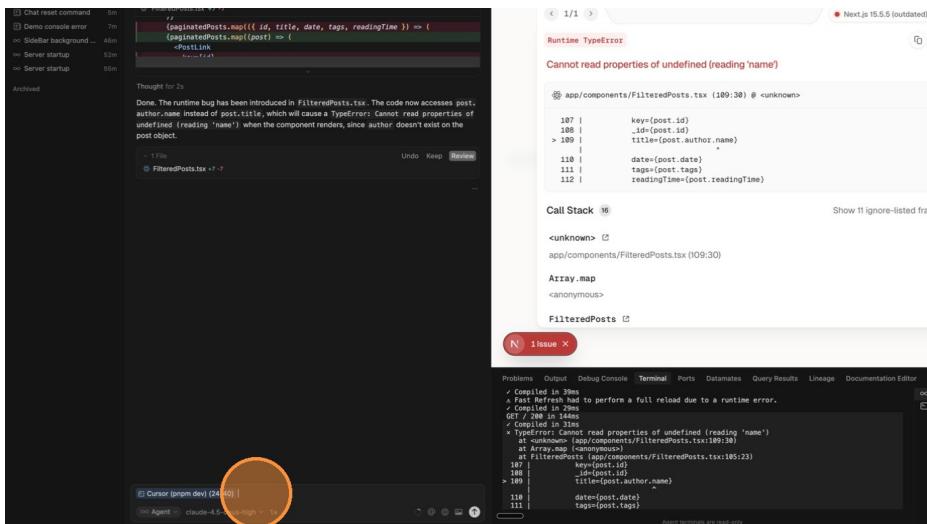
Error added to chat context (Cmd+L):



The screenshot shows a developer's terminal window with several tabs open. One tab is titled 'FilteredPosts.tsx' and contains code. Another tab shows a runtime error message: 'Done. The runtime bug has been introduced in FilteredPosts.tsx. The code now accesses post.author.name instead of post.title, which will cause a TypeError: Cannot read properties of undefined (reading 'name') when the component renders, since author doesn't exist on the post object.' A cursor is highlighted over the word 'name' in the error message.

Error in chat context

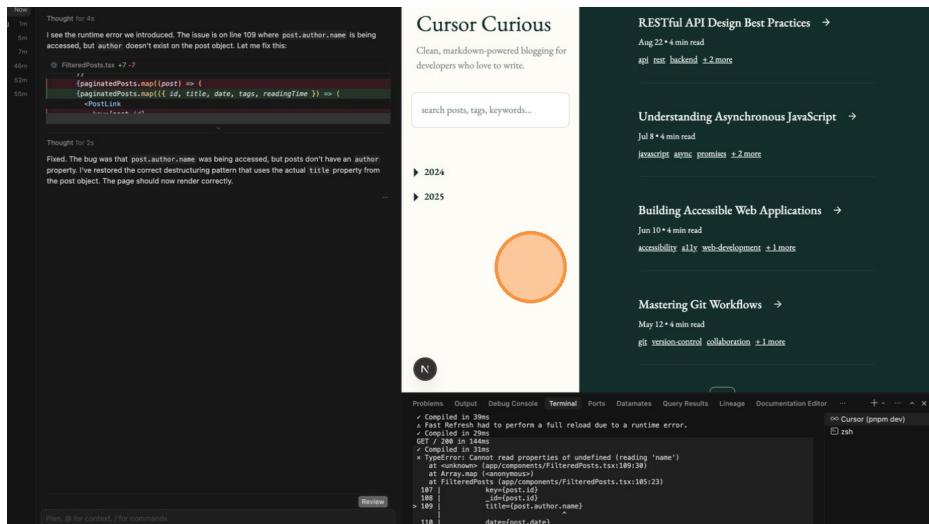
File location highlighted:



The screenshot shows a developer's terminal window with several tabs open. One tab is titled 'FilteredPosts.tsx' and contains code. Another tab shows a runtime error message: 'Done. The runtime bug has been introduced in FilteredPosts.tsx. The code now accesses post.author.name instead of post.title, which will cause a TypeError: Cannot read properties of undefined (reading 'name') when the component renders, since author doesn't exist on the post object.' A cursor is highlighted over the file path 'app/components/FilteredPosts.tsx' in the terminal output.

File location shown

Submit to have the agent fix it:



Submit fix request

3.10 Using Documentation

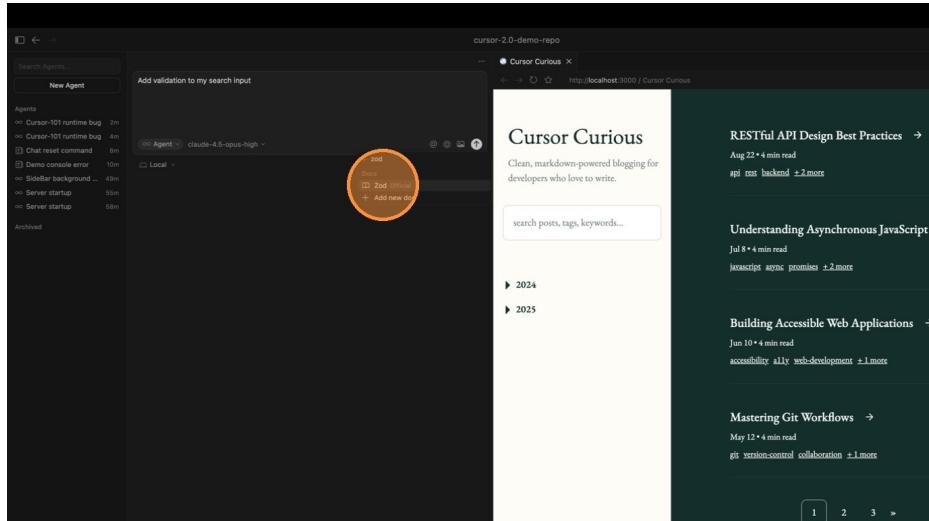
Reference up-to-date library documentation so the agent uses current APIs and patterns.

Built-in Docs

Popular libraries are pre-indexed. Type `@docs` and search:

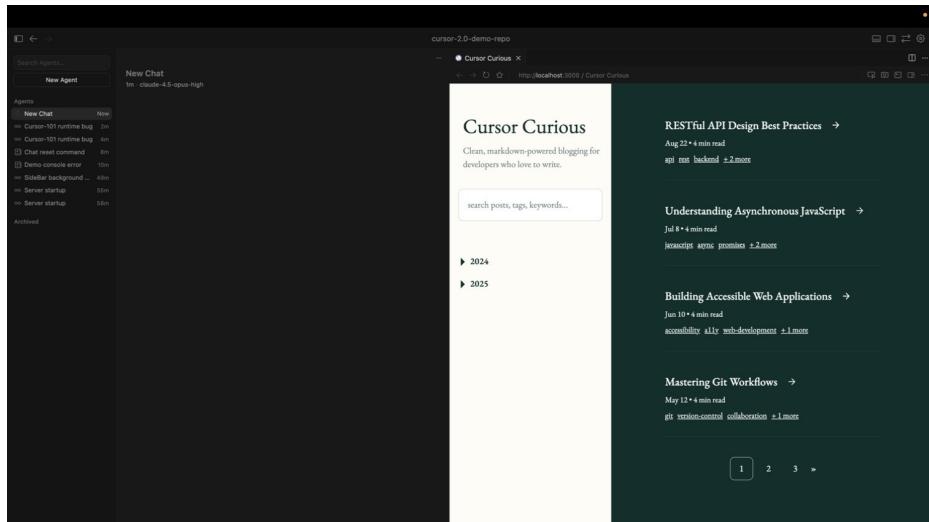
- Add validation with Zod

Search for documentation:



Search docs

Select from available documentation:

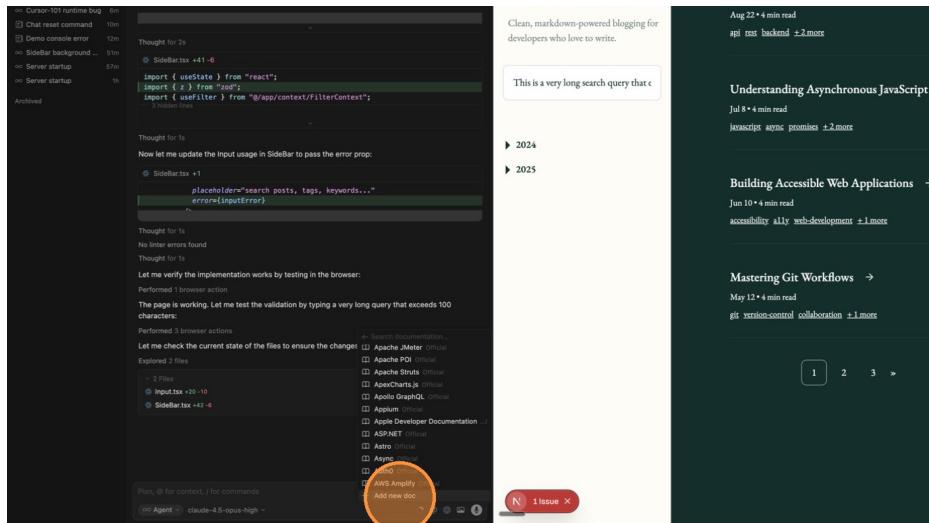


Select Zod docs

Adding Custom Docs

1. Click **Add Context > Docs**
2. Scroll to bottom, click **Add new doc**
3. Paste the documentation URL
4. Cursor will index it for future use

Click Add new doc:



Add new doc option

Paste documentation URL:

The screenshot shows the Valibot API reference documentation. The main title is "API reference". Below it, there's a section titled "Schemas" with a list of schema types: any, array, bigint, blob, boolean, custom, date, enum, exactOptional, file, and function. To the right, there's a sidebar titled "On this page" with links to Schemas, Methods, Actions, Storages, Utils, Async, and Types.

Paste doc URL

URL pasted and ready to index:

The screenshot shows the Cursor interface with a document titled "Search input validation with Zod". The document content includes code snippets and comments. A modal window is open in the center, prompting the user to "Add docs" with the URL "https://valibot.dev/api/". The "Add new doc" button is highlighted with a red circle.

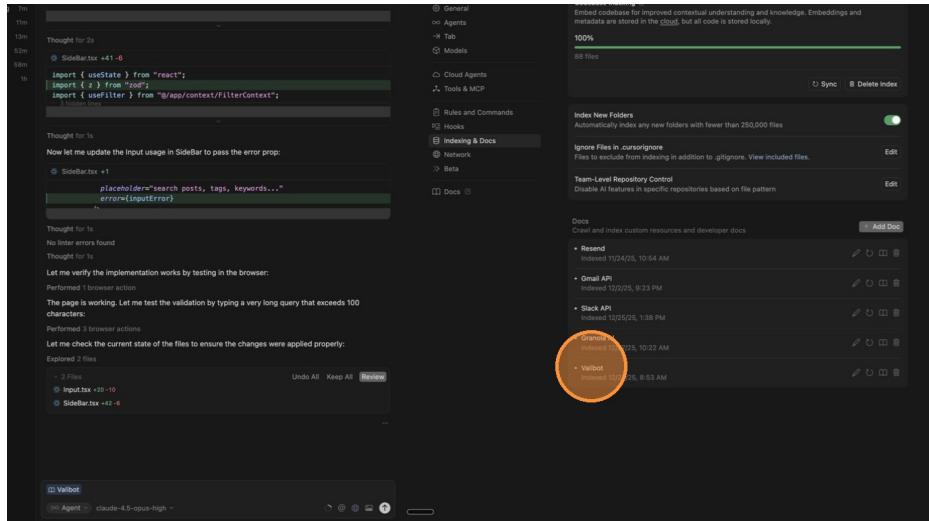
URL ready

Documentation indexing:

The screenshot shows the Cursor interface with the same document as the previous screenshot. A modal window is open, showing the "Add new doc" form with the URL "https://valibot.dev/api/" entered. A "Confirm" button is visible at the bottom right of the modal. The "Add new doc" button is highlighted with a red circle.

Indexing docs

Documentation indexed and available:



Docs indexed

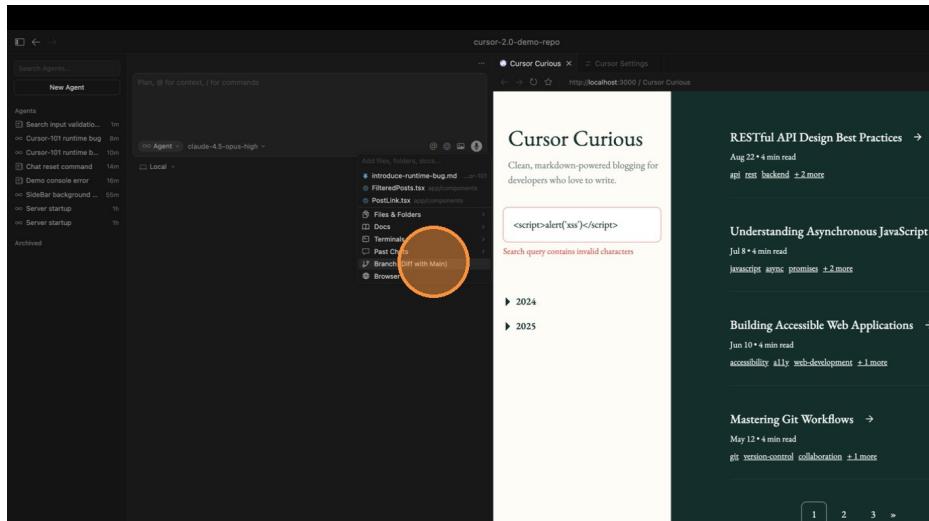
3.11 Git Diff Context

Review changes against your main branch and generate commit messages.

Use `@Git (Diff with Main Branch)` to: - Understand what your feature branch changed - Generate commit messages based on actual changes - Get an overview of your current work status

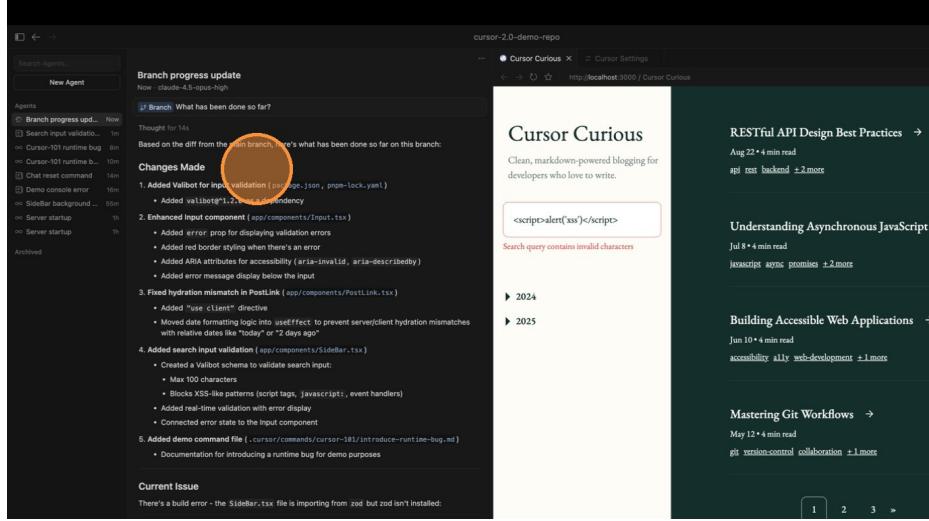
- Explain changes against main

Select Git Diff context:



Git diff context

Agent explains the changes:

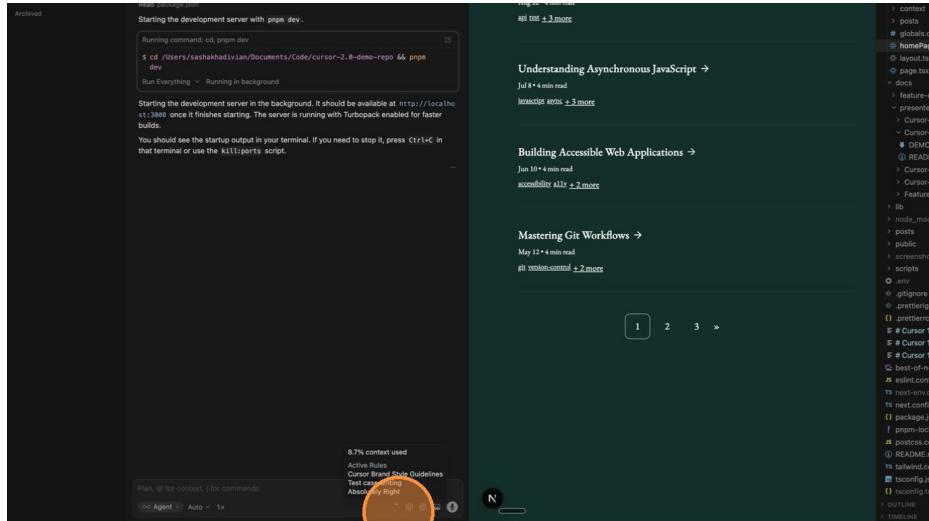


Changes explained

3.12 Context Window Management

Monitor your agent's context usage to ensure optimal performance.

View context usage indicator:

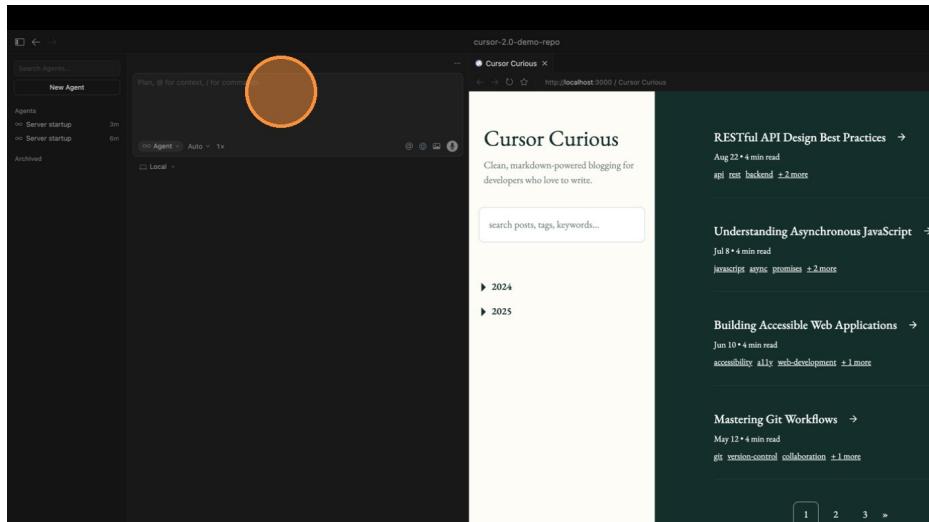


Context window indicator

As conversations grow, the context window fills up. When it reaches capacity: 1. Cursor summarizes the conversation 2. Resets with the summary as context 3. Continues the task seamlessly

Best Practice: Use atomic tasks—one focused task per agent session. Start new agents for new tasks to keep context clean.

Start a new agent:

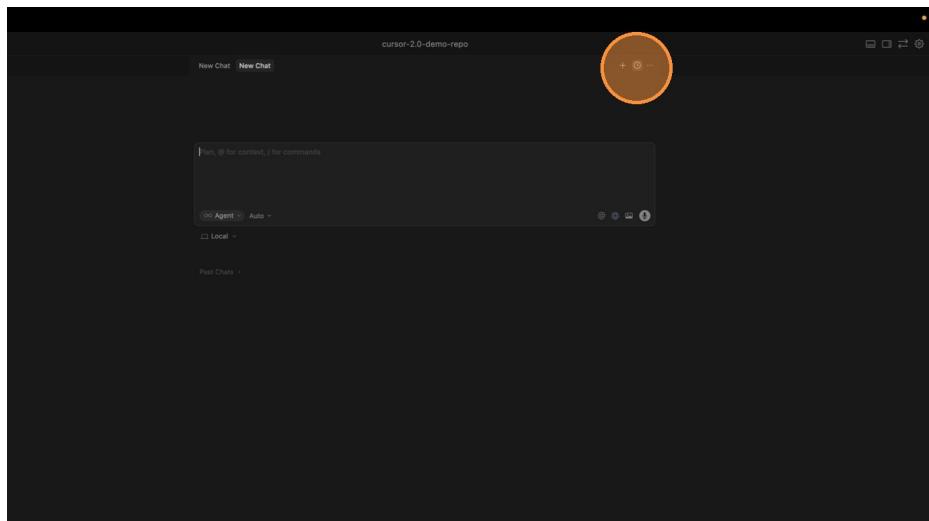


New agent button

3.13 Chat History

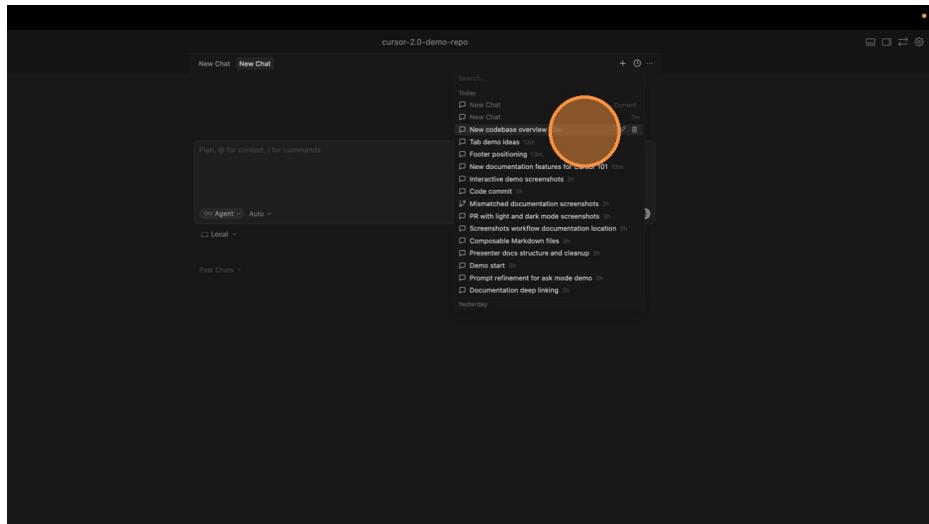
Access previous conversations to continue work or reference past solutions.

Click Chat History:



Chat history button

Browse past conversations:



Chat history list

3.14 Demo: Implementing Dark Mode

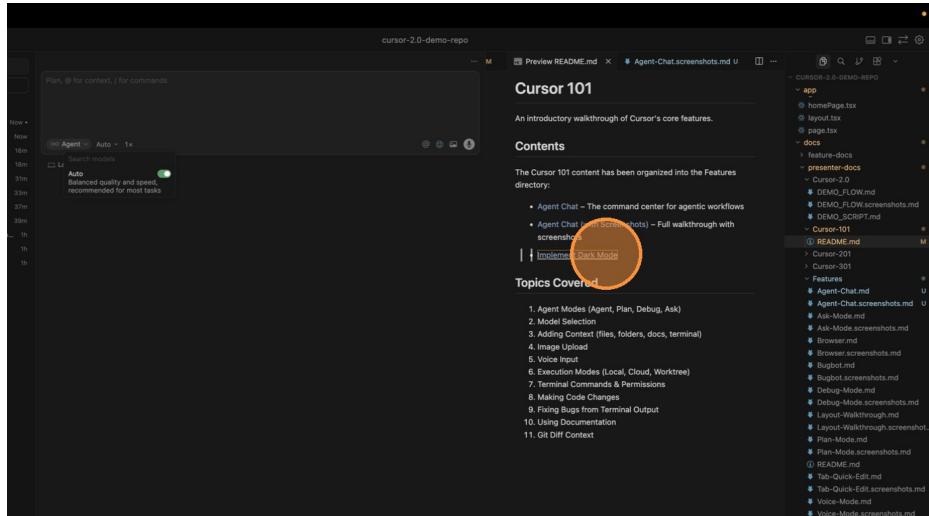
Walk through implementing a complete feature using Agent Chat.

This demo shows the full workflow: clicking a deep link, selecting a model, running the agent, and reviewing changes.

Steps

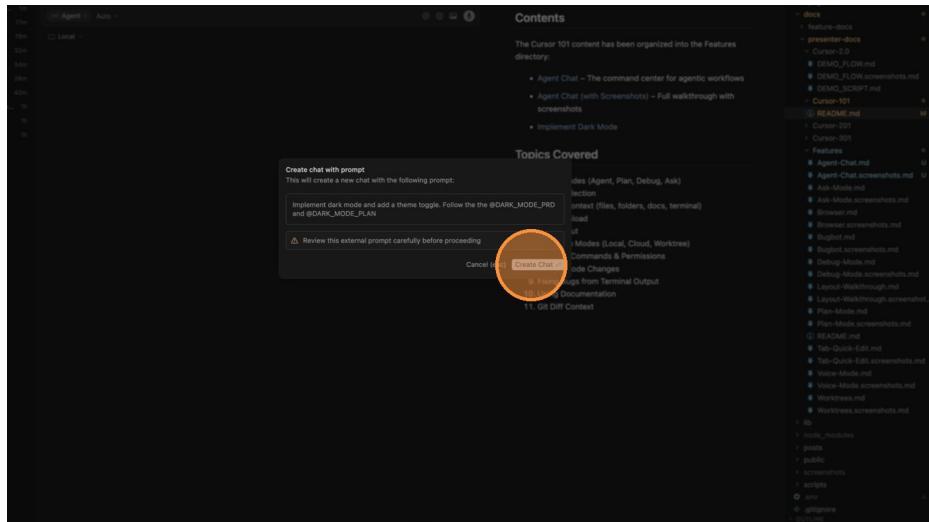
- Implement Dark Mode

Click the deep link to open the prompt:



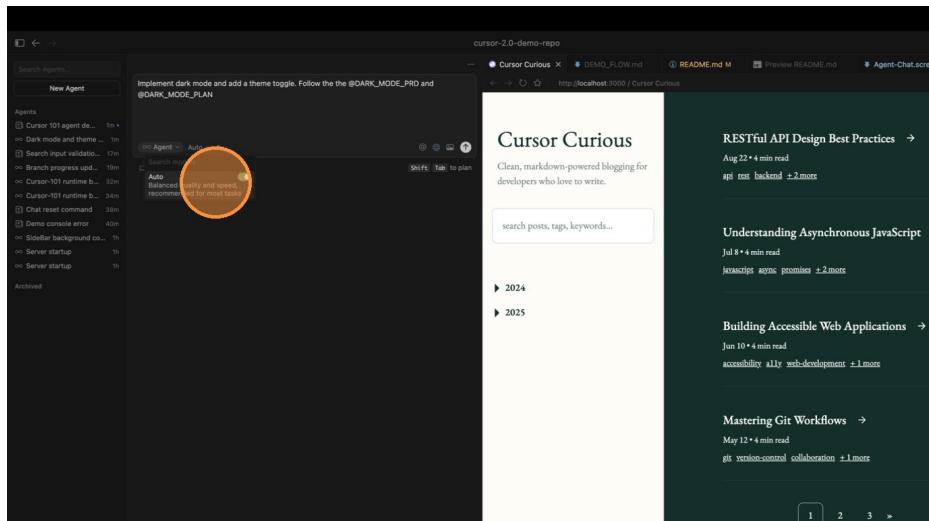
Click Implement Dark Mode

Click Create Chat:



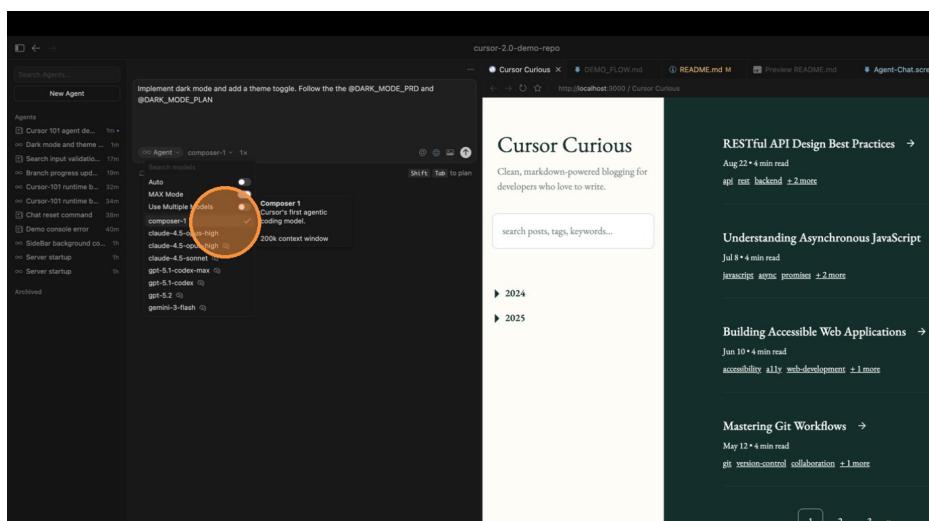
Create Chat dialog

Click the model dropdown:



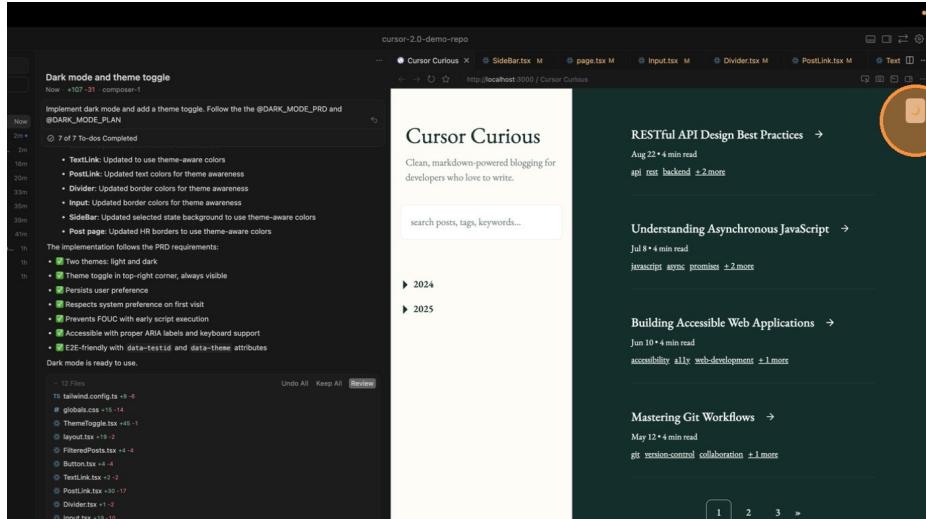
Model dropdown

Select composer-1 for faster results:



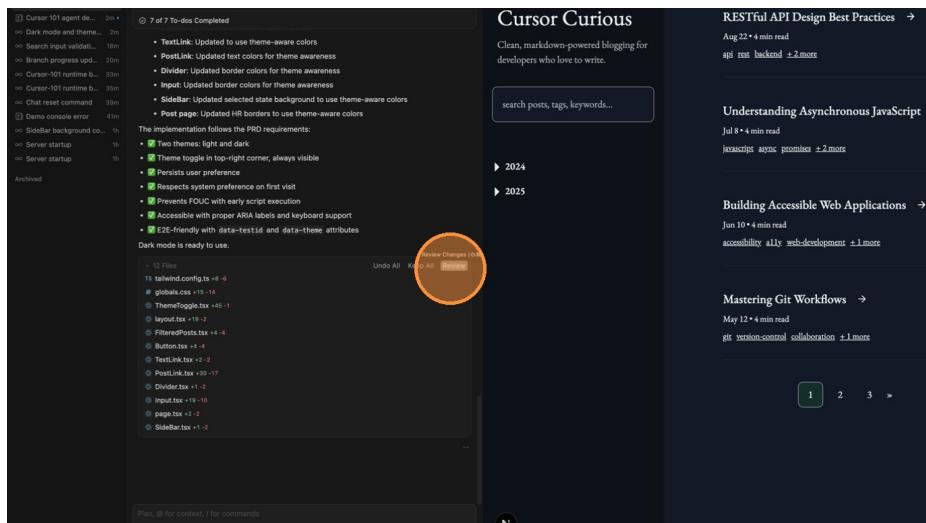
Select composer-1

Agent implements the feature:



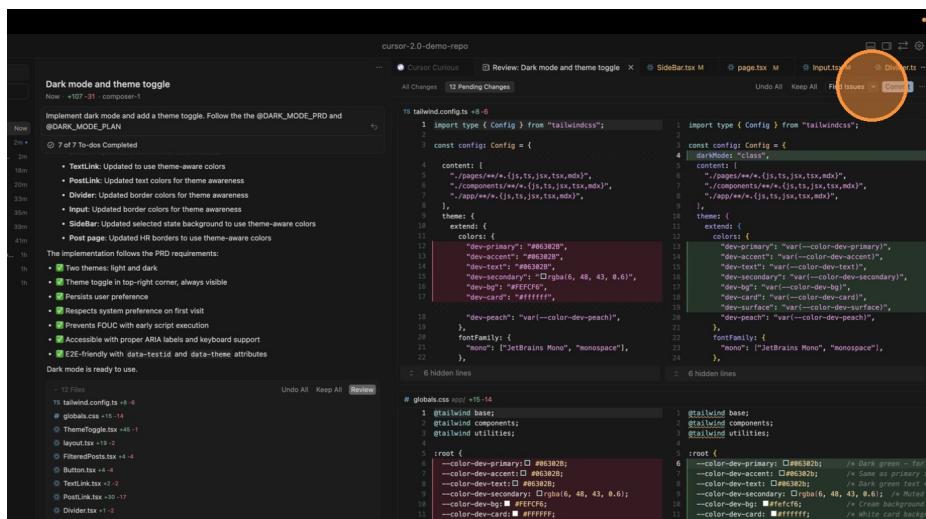
Agent working

Click Review to see all changes:



Click Review

Review changes across files:



Review scope

What to Highlight

- Deep links pre-fill prompts for consistent demos
 - Model selection affects speed and quality
 - Agent reads existing code to understand patterns
 - Review mode shows all changes across files
 - Changes can be accepted or reverted individually
-

Quick Reference

Action	Shortcut/Command
Open Agent Chat	Cmd+L
Quick Edit	Cmd+K
Accept Tab suggestion	Tab
Reject Tab suggestion	Escape
Chat History	Opt+Cmd+'
New Chat	Click “New Chat” button
Add Selection to Chat	Select text + Cmd+L
Voice Input	Click microphone button
Start demo server	<u>/start-demo</u>
Reset workspace	<u>/reset</u>

Next Steps

After completing Cursor 101, explore:

- **Cursor 2.0 Demo** – Advanced features like Browser, Worktrees, Bugbot
- **Plan Mode** – Detailed implementation planning
- **Debug Mode** – Hypothesis-driven debugging
- **Browser** – Built-in browser testing