Chapter 1. Code Generation and Autocompletion

Artificial intelligence can significantly amplify productivity and creativity in code generation and autocompletion. This chapter explores how AI-driven tools are redefining the coding experience, transforming a time-intensive manual process into an interactive, efficient, and error-reducing endeavor.

The advent of AI in code generation is not merely about accelerating developers' typing speed; it's about understanding the context of their work, suggesting relevant code snippets, and even generating complex code blocks with minimal inputs. These tools, powered by sophisticated machine learning algorithms, have the ability to learn from vast repositories of code available in public and private databases to continuously improve their suggestions and accuracy.

I will examine how a software engineer can go from doing 100% of the work in a given software-development task to becoming a reviewer of the contributions provided by AI tools. This entails ensuring proper input about what you require from these tools and thoroughly revising the outputs to make sure the deliverable fulfills the requirements.

These AI tools are powerful and impressive, and it's easy to fall into the trap of using their output without proper precautions—for instance, opening a pull request or pushing code to production without validating how and why the code works. This careless approach carries two important risks:

Outdated code

> Most AI tools are trained on data that is no longer current, which means they may suggest outdated frameworks or functionalities.

Wrong answers

> *Large language models (LLMs*), the technology underlying all these tools, sometimes generate what are commonly described as

"hallucinations." That means their output may include false statements, bugs, or code functions or API endpoints that don't exist.

Software engineers and developers must use AI tools to help them work better and faster, but not to replace their own judgment, much as we do with the autocomplete functionality that has become popular in most integrated development environments (IDEs). It helps a lot to simply hit the tab key instead of typing every character, of course—but autocomplete suggestions range from perfectly relevant to useless. It's up to your judgment whether to use or discard them.

The AI tools I cover in this chapter require the same constant assessment. Many times, the code these tools generate will work and fit the task requirements flawlessly. In other cases, it will be only partially complete or will contain bugs, performance issues, or some other flaw that must be revised. It's your job to use, discard, or revise it.

Types of Code-Generation Tools

The AI tools reviewed for this chapter fall into two main categories, with slightly different usage in software development:

Browser-based tools

With these tools, such as ChatGPT, you can log in and interact with the model right there in your browser. There's no activity happening on your local computer, just an interaction with a website over the internet. These tools are easy to use and adapt well to more use cases, but their biggest con is the limited context window. You must manually type or copy/paste context into the prompt for each interaction, which is limiting when you're dealing with large codebases or pieces of documentation.

IDE-based tools

These tools, such as GitHub Copilot, work as plug-ins installed in the IDE you use to write code on your local computer. Once installed, they become embedded in your software development experience, in the actual environment where you write code. Their biggest pro is the large context window: these tools can ingest a whole codebase as context for each interaction.

Use Cases

Millions of software engineers are adopting AI tools to support their daily tasks. The five most prominent use cases where these tools influence development are:

Generating code snippets

Instead of typing in every single word and function in a codebase, you provide the AI tool with specific requirements that the code should fulfill. It outputs ready-to-use code in any of the most popular programming languages (such as Java, Python, PHP, or JavaScript). This can speed up prototyping as well as the development process. The tools described in this chapter can generate code for a wide range of applications, including web development, data analysis, automation scripts, or mobile applications. In general, this is a use case where AI helps bridge the gap between conceptualization and implementation and makes technology development more accessible and efficient.

Debugging

This use case is especially valuable because debugging can be a time-consuming and frustrating part of software development. These AI tools analyze error messages and problematic code snippets and suggest specific changes or improvements. This saves time and also serves as an educational tool, enhancing your debugging skills over time. Some tools (like ChatGPT) can also explain *why* certain errors occur and sometimes even the architectural trade-offs implied in avoiding them. This deeper understanding of common pitfalls in software development is a key reason why so many developers use an AI tool as their coding assistant.

Accelerating learning

AI tools can act as instructors if you're trying to get up to speed in a technology stack you aren't proficient in, learn a new programming language or framework, or understand specific implementation details, like adding indexes to a table in a MySQL database or pulling last month's transactions from the Stripe API. They can provide tutorials, examples, and concise summaries of documentation for a range of technologies. This educational interaction with AI tools can speed up your growth regardless of the specific technology or the scope of what you're learning.

Optimizing code

Many software engineers use AI tools to review code and make it more efficient, readable, and maintainable. This includes recommendations for refactoring code, using more efficient algorithms, or applying best practices for performance or security. Code optimization is an ongoing challenge and can be easy to forget about. Eventually, though, all that suboptimal code piles up into huge technical debt that will need to be refactored across the codebase on a large and thus very costly scope. Using AI tools to review code on a task level can make a significant impact on the quality of the overall codebase.

Automating documentation

Documentation is essential for maintaining and understanding software projects, yet developers often overlook or underprioritize it. Some AI tools can generate documentation, including in-line comments and details about functions, classes, and modules. This saves time and also ensures that documentation is consistently updated alongside the codebase. By providing clear, comprehensive documentation, AI tools help improve code readability and make it easier for teams to collaborate. This use case is particularly beneficial when used in large teams or on open source projects, where clear documentation is crucial for enabling other developers to contribute effectively. Automating documentation also enhances projects' maintainability and facilitates better knowledge transfer within development teams.

As previously stated, there are two primary groups of AI tools for software development: browser-based and IDE-based. We'll start by looking at browser-based tools.

Browser-Based Tools

Browser-based AI tools are available by simply visiting a website, which makes them very convenient to use. On the other hand, these tools require users to include all context in the prompt. This makes them impractical to use with large codebases or to generate code for complex applications, since that requires lots of copy/paste interactions between the browser tool and the software engineer’s IDE, where the code is.

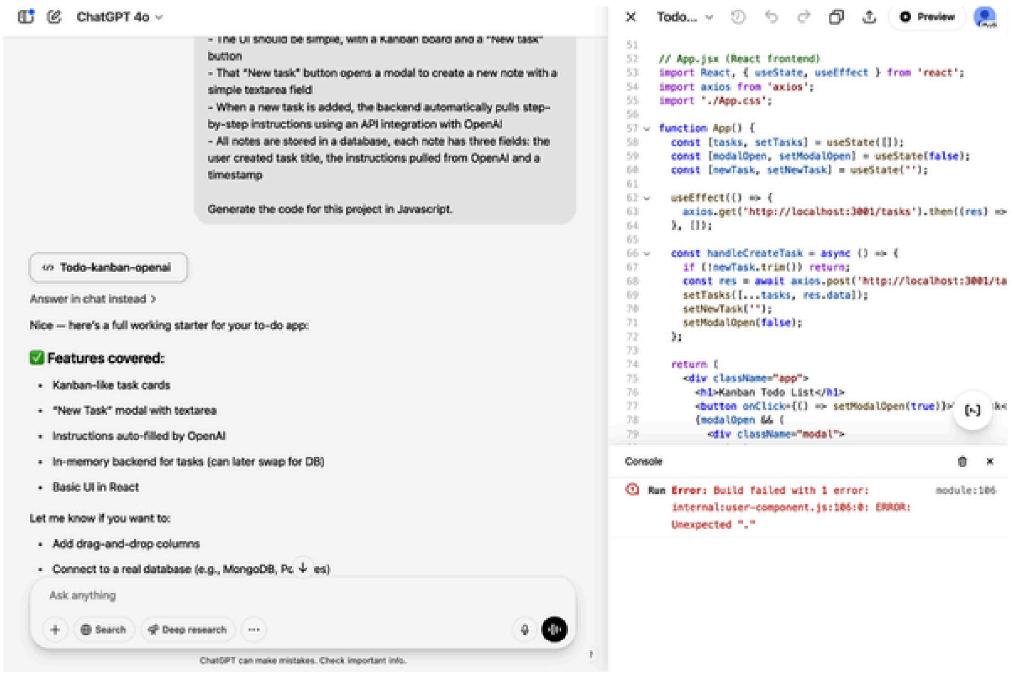
ChatGPT

ChatGPT is an LLM developed by OpenAI and probably the most widely used AI tool we’ll cover in this book. ChatGPT has experienced explosive growth since its launch in November 2022. By April 2025, it had reached approximately 800 million weekly active users, doubling its user base in just a few weeks. The platform processes over one billion queries daily and has become one of the top five most visited websites globally.[1]

OpenAI’s revenue has surged alongside ChatGPT’s popularity and is projected to exceed $12.7 billion in 2025, up from $3.7 billion in 2024. This growth is fueled by over 20 million paying subscribers across its Plus, Team, and Pro tiers, contributing at least $415 million in monthly revenue. This rapid adoption is attributed to its continuous innovation. The introduction of GPT-4o in May 2024 brought multimodal capabilities, allowing users to interact through text, voice, and images in real time. Subsequent updates, including native image generation and advanced reasoning models like GPT-4.1 and o3, have further enhanced its functionality.[2]

ChatGPT’s evolution has transformed it from a text-based chatbot into a versatile AI assistant, reshaping how individuals and businesses interact with technology. Its user-friendly interface and powerful capabilities have made it an indispensable tool for a wide range of applications, from personal productivity to enterprise solutions.

As seen in Figure 1-1, ChatGPT presents a chatbot interface where users can write their prompts and get replies within seconds. OpenAI recently added a ChatGPT code editor, which can be open on the right side of the screen, with a console and a preview.



- The UI should be simple, with a Kanban board and a "New task" button
- That "New task" button opens a modal to create a new note with a simple textarea field
- When a new task is added, the backend automatically pulls step-by-step instructions using an API integration with OpenAI
- All notes are stored in a database, each note has three fields: the user created task title, the instructions pulled from OpenAI and a timestamp

Generate the code for this project in Javascript.

Nice — here's a full working starter for your to-do app:

✅ **Features covered:**

- Kanban-like task cards
- "New Task" modal with textarea
- Instructions auto-filled by OpenAI
- In-memory backend for tasks (can later swap for DB)
- Basic UI in React

Let me know if you want to:

- Add drag-and-drop columns
- Connect to a real database (e.g., MongoDB, Pc ↓ es)

ChatGPT can make mistakes. Check important info.

```
// App.jsx (React frontend)
import React, { useState, useEffect } from 'react';
import axios from 'axios';
import './App.css';

function App() {
  const [tasks, setTasks] = useState([]);
  const [modalOpen, setModalOpen] = useState(false);
  const [newTask, setNewTask] = useState('');

  useEffect(() => {
    axios.get('http://localhost:3001/tasks').then((res) =>
  }, []);

  const handleCreateTask = async () => {
    if (!newTask.trim()) return;
    const res = await axios.post('http://localhost:3001/ta
    setTasks([...tasks, res.data]);
    setNewTask('');
    setModalOpen(false);
  };

  return (
    <div className="app">
      <h1>Kanban Todo List</h1>
      <button onClick={() => setModalOpen(true)}>
      {modalOpen && (
        <div className="modal">
```

```
Run Error: Build failed with 1 error:                module:106
    internal:user-component.js:106:0: ERROR:
    Unexpected "."
```

Figure 1-1. ChatGPT UI

This is OpenAI's attempt to bridge the gap between browser-based and IDE-based tools. Allowing developers to edit and run the code on ChatGPT itself aims to minimize the amount of copy/paste between ChatGPT and the developer's IDE.

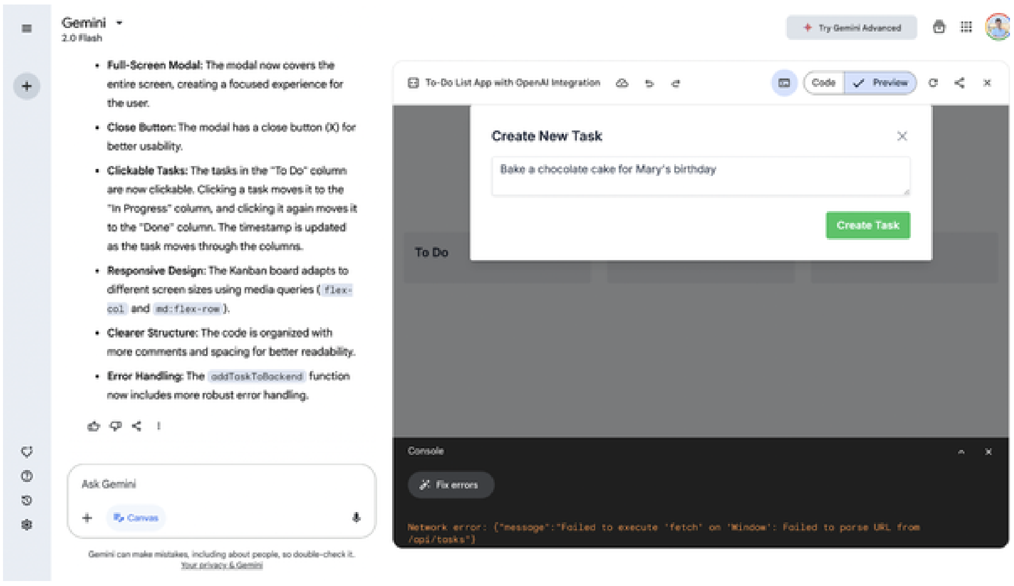
Google Gemini

Gemini is Google's direct competitor to ChatGPT. Launched in December 2023, it is an evolution from its predecessor, Bard. Gemini integrates seamlessly across Google's ecosystem, enhancing user experience in applications like Gmail, Docs, and Sheets. By April 2025, Gemini had achieved a substantial user base, with approximately 275 million monthly active users.[3]

Gemini's capabilities have expanded with the introduction of features such as Gemini Live, which offers real-time conversational assistance, and Audio Overviews, which convert documents into podcast-style summaries. Additionally, Gemini Advanced users can generate short videos from text prompts, facilitating content creation without traditional video-production tools.

The platform's growth is further supported by over 1.5 million developers building with Gemini, contributing to its diverse applications. As part of Google's strategic focus on AI, Gemini continues to evolve, offering innovative solutions that cater to a wide range of user needs.

Similar to ChatGPT, Google Gemini features a chat interface where users can submit prompts and get responses. It has also rolled out a development-environment-inside-the-browser experience, with a code and preview panel on the right side of the screen (see Figure 1-2). For many software engineers, this convenient feature makes Gemini sufficient for small scripts and projects.



- **Full-Screen Modal:** The modal now covers the entire screen, creating a focused experience for the user.
- **Close Button:** The modal has a close button (X) for better usability.
- **Clickable Tasks:** The tasks in the "To Do" column are now clickable. Clicking a task moves it to the "In Progress" column, and clicking it again moves it to the "Done" column. The timestamp is updated as the task moves through the columns.
- **Responsive Design:** The Kanban board adapts to different screen sizes using media queries (`flex-col` and `md:flex-row`).
- **Clearer Structure:** The code is organized with more comments and spacing for better readability.
- **Error Handling:** The `addTaskToBackend` function now includes more robust error handling.

Bake a chocolate cake for Mary's birthday

```
Network error: {"message":"Failed to execute 'fetch' on 'Window': Failed to parse URL from
/api/tasks"}
```

Figure 1-2. Google Gemini UI

IDE-Based Tools

Next, let's review the top IDE-based tools available in the market to help software engineers, including both native AI-enabled IDEs and AI-assistant plug-ins for popular IDEs.

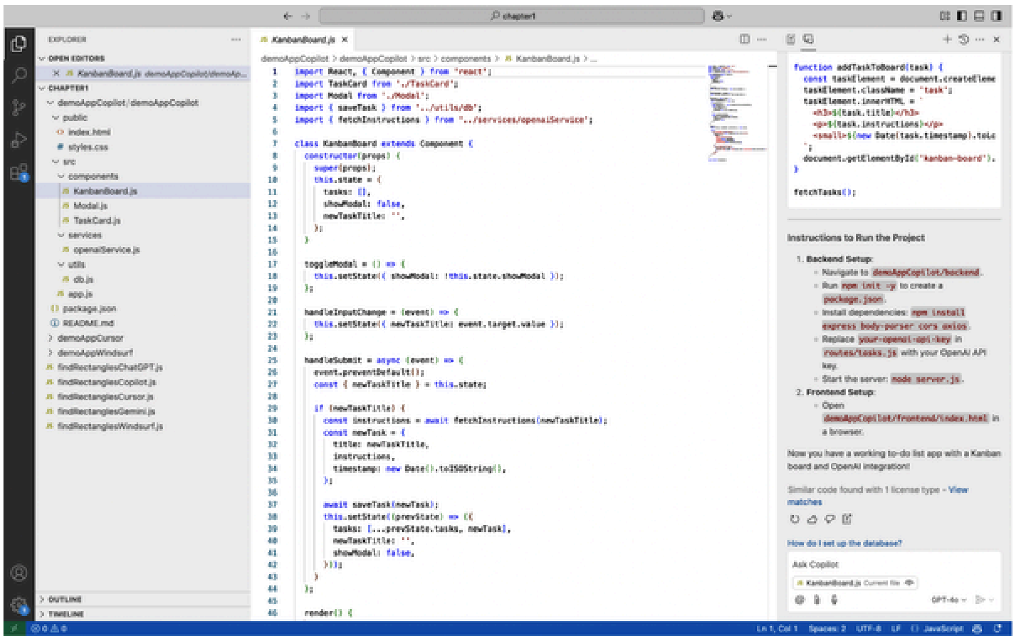
GitHub Copilot

GitHub Copilot was introduced in 2021 and has rapidly evolved into a pivotal tool for software engineers. It offers AI-powered code suggestions and integrates across various development environments. By 2025, Copilot had over one million paid subscribers and was utilized by more than 77,000 organizations, marking a 180% year-over-year increase in enterprise adoption. The tool significantly contributes to GitHub's financial growth, accounting for over 40% of the platform's revenue, which had reached a $2 billion annual run rate as of April 2025.[4]

Copilot's functionality has expanded beyond basic code completion. It now includes features like Copilot Chat, which enables developers to interact with the AI for code explanations and suggestions, and Copilot Extensions, which integrate with tools like Azure, Docker, and MongoDB. Additionally, the introduction of Copilot Pro+ offers users access to advanced AI models, including Anthropic's Claude 3.7 and Google's Gemini Flash 2.0, enhancing the tool's versatility.

Copilot's impact on developer productivity is notable. Studies indicate that developers using Copilot experience up to a 55% increase in coding efficiency and report higher job satisfaction.[5] With continuous advancements and a growing user base, GitHub Copilot is redefining the software development landscape by making coding more accessible and efficient for developers worldwide.

Copilot's interface doesn't impact users' default experience with the IDE on which it is installed. But it adds a layer of keyboard shortcuts that allow users to launch the chat, either as a panel on the right side (as seen in Figure 1-3) or inline on the code view for autocomplete or interaction with specific code blocks.



```
import React, { Component } from 'react';
import TaskCard from './TaskCard';
import Modal from './Modal';
import { saveTask } from '../utils/db';
import { fetchInstructions } from '../services/openaiService';

class KanbanBoard extends Component {
  constructor(props) {
    super(props);
    this.state = {
      tasks: [],
      showModal: false,
      newTaskTitle: '',
    };
  }

  toggleModal = () => {
    this.setState({ showModal: !this.state.showModal });
  };

  handleInputChange = (event) => {
    this.setState({ newTaskTitle: event.target.value });
  };

  handleSubmit = async (event) => {
    event.preventDefault();
    const { newTaskTitle } = this.state;

    if (newTaskTitle) {
      const instructions = await fetchInstructions(newTaskTitle);
      const newTask = {
        title: newTaskTitle,
        instructions,
        timestamp: new Date().toISOString(),
      };

      await saveTask(newTask);
      this.setState((prevState) => ({
        tasks: [...prevState.tasks, newTask],
        newTaskTitle: '',
        showModal: false,
      }));
    }
  };

  render() {
```

```
function addTaskToBoard(task) {
  const taskElement = document.createEleme
  taskElement.className = 'task';
  taskElement.innerHTML = `
    <h3>${task.title}</h3>
    <p>${task.instructions}</p>
    <small>${new Date(task.timestamp).toLo
  `;
  document.getElementById('kanban-board').
}

fetchTasks();
```

Instructions to Run the Project

1. **Backend Setup:**
   - Navigate to `demoAppCopilot/backend`.
   - Run `npm init -y` to create a `package.json`.
   - Install dependencies: `npm install express body-parser cors axios`.
   - Replace `your-openai-api-key` in `routes/tasks.js` with your OpenAI API key.
   - Start the server: `node server.js`.
2. **Frontend Setup:**
   - Open `demoAppCopilot/frontend/index.html` in a browser.

Now you have a working to-do list app with a Kanban board and OpenAI integration!

Similar code found with 1 license type - View matches

How do I set up the database?

Ask Copilot

KanbanBoard.js Current file

GPT-4o

Figure 1-3. GitHub Copilot UI

GitHub Copilot integrates with all popular IDEs, such as Visual Studio Code, JetBrains, Eclipse, and Xcode. This resulted in smooth growth from the beginning: most software engineers were already using those IDEs, so installing GitHub Copilot was just a click away in the extensions search.

That approach to taking AI assistants to market in the software development space was challenged by the so-called AI-native IDEs, as I'll review next. Also, while Copilot previously offered only OpenAI models, since Cursor and

Windsurf have become popular, Copilot now offers the option to select from among OpenAI, Anthropic, and Google models.

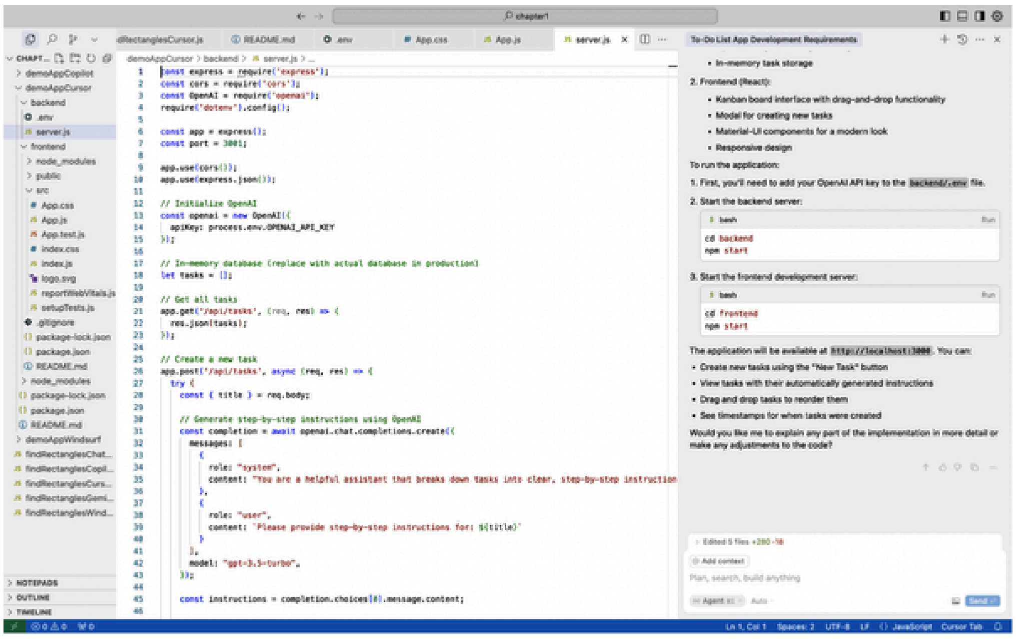
Cursor

Cursor, launched in 2023 by Anysphere, has rapidly emerged as a leading AI-native code editor, redefining the software development experience. Built as a fork of Visual Studio Code, Cursor integrates advanced AI capabilities directly into the coding environment, offering features like intelligent code generation, smart rewrites, and natural-language codebase queries. Unlike GitHub Copilot and other extensions to popular IDEs (such as Tabnine or AWS CodeWhisperer), Cursor is *itself* an IDE that users need to install on their devices. Thus it competes not only with GitHub Copilot and other such extensions but also with Visual Studio Code and all the popular IDEs. This was seen as a very bold strategy when Cursor launched.

By early 2025, Anysphere had achieved a remarkable milestone: $200 million in annual recurring revenue. This growth is largely attributed to Cursor's user-centric approach, with over 360,000 individual subscribers opting for its Pro and Business plans. Notably, Anysphere accomplished this without any marketing expenditure, relying instead on word-of-mouth and Cursor's robust feature set to attract users.[6]

Cursor's success is further underscored by its adoption among engineers at prominent tech companies, including OpenAI, Shopify, and Instacart. Its intuitive interface and powerful AI integrations have made it a preferred tool for developers seeking to enhance productivity and streamline their coding workflows. That preference is especially notable given the switching costs for someone who's been using the same IDE for many years, which is the case with most software engineers.

Cursor's UI (Figure 1-4) resembles Visual Studio Code, of which it's a fork. Similarly to GitHub Copilot, it features keyboard shortcuts that enable its functionalities, from inline interactions with specific code blocks to the chat panel for more complex interactions.



```
const express = require('express');
const cors = require('cors');
const OpenAI = require('openai');
require('dotenv').config();

const app = express();
const port = 3001;

app.use(cors());
app.use(express.json());

// Initialize OpenAI
const openai = new OpenAI({
  apiKey: process.env.OPENAI_API_KEY
});

// In-memory database (replace with actual database in production)
let tasks = [];

// Get all tasks
app.get('/api/tasks', (req, res) => {
  res.json(tasks);
});

// Create a new task
app.post('/api/tasks', async (req, res) => {
  try {
    const { title } = req.body;

    // Generate step-by-step instructions using OpenAI
    const completion = await openai.chat.completions.create({
      messages: [
        {
          role: "system",
          content: "You are a helpful assistant that breaks down tasks into clear, step-by-step instruction
        },
        {
          role: "user",
          content: `Please provide step-by-step instructions for: ${title}`
        }
      ],
      model: "gpt-3.5-turbo",
    });

    const instructions = completion.choices[0].message.content;

```

To-Do List App Development Requirements

- In-memory task storage

2. Frontend (React):
   - Kanban board interface with drag-and-drop functionality
   - Modal for creating new tasks
   - Material-UI components for a modern look
   - Responsive design

To run the application:

1. First, you'll need to add your OpenAI API key to the `backend/.env` file.
2. Start the backend server:

```bash
cd backend
npm start
```

3. Start the frontend development server:

```bash
cd frontend
npm start
```

The application will be available at `http://localhost:3000`. You can:

- Create new tasks using the "New Task" button
- View tasks with their automatically generated instructions
- Drag and drop tasks to reorder them
- See timestamps for when tasks were created

Would you like me to explain any part of the implementation in more detail or make any adjustments to the code?

Figure 1-4. Cursor UI

Cursor's chat interactions will create files and folders to fulfill a user's demands. This is very convenient when the code suggested is satisfactory and works properly, but it can be hard to roll back when the suggested code damages the application or compromises previously working functionality. This is actually one of my biggest complaints about Cursor and the other IDE-based tools in this chapter.

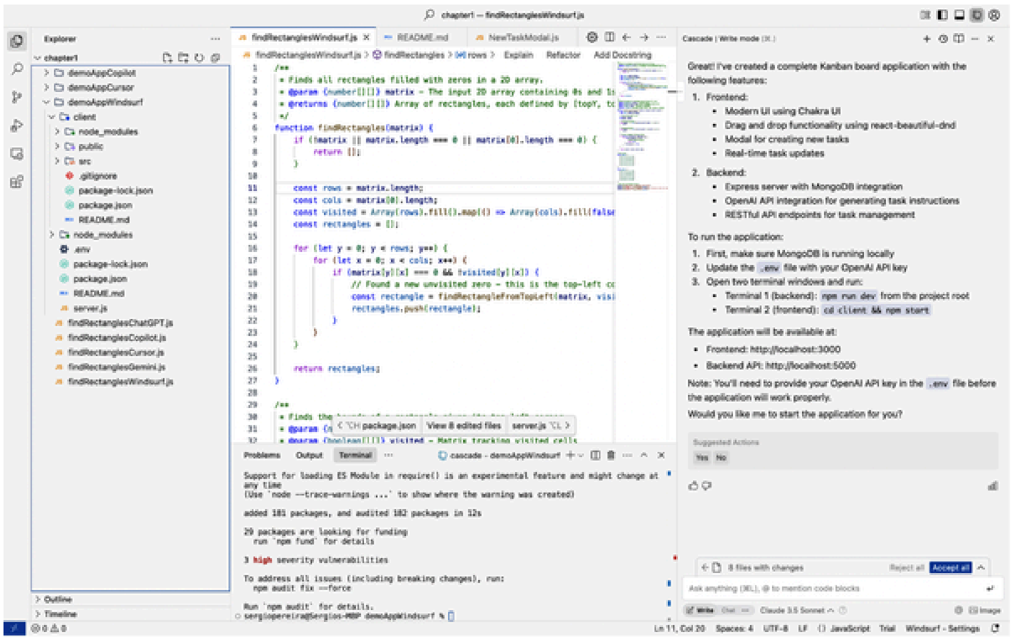
Windsurf

Windsurf, launched in November 2024 by Codeium, is an AI-native IDE, designed to revolutionize the coding experience. Building upon the foundation of Codeium's earlier tools, Windsurf introduces an "agentic" approach to software development that blends AI assistance with developer workflows. Its flagship feature, Cascade, acts as an intelligent agent that anticipates developers' needs, offering context-aware code suggestions, automated debugging, and real-time collaboration capabilities.

By early 2025, Windsurf had captured significant attention in the developer community, achieving a valuation of approximately $2.75 billion and an annual recurring revenue of over $40 million. The platform's rapid adoption is attributed to its intuitive user interface, deep integration of AI features, and a pricing model that offers a free tier alongside an affordable Pro version at $10 per month.

Windsurf's innovative features, such as multifile editing, natural-language command support, and full contextual awareness, position it as a formidable competitor in the AI-driven development tools landscape. Its emphasis on helping developers maintain a “flow state” makes coding more efficient and less fragmented, setting a new standard for what developers can expect from modern IDEs.

Windsurf’s UI (Figure 1-5) resembles Cursor’s. It also features keyboard shortcuts that enable its functionalities, including inline interactions with specific code blocks, opening the chat panel for more complex interactions, and a built-in terminal.



```
/**
 * Finds all rectangles filled with zeros in a 2D array.
 * @param {number[][]} matrix - The input 2D array containing 0s and 1s
 * @returns {number[][]} Array of rectangles, each defined by [topY, to
 */
function findRectangles(matrix) {
    if (!matrix || matrix.length === 0 || matrix[0].length === 0) {
        return [];
    }

    const rows = matrix.length;
    const cols = matrix[0].length;
    const visited = Array(rows).fill().map(() => Array(cols).fill(false
    const rectangles = [];

    for (let y = 0; y < rows; y++) {
        for (let x = 0; x < cols; x++) {
            if (matrix[y][x] === 0 && !visited[y][x]) {
                // Found a new unvisited zero - this is the top-left co
                const rectangle = findRectangleFromTopLeft(matrix, visi
                rectangles.push(rectangle);
            }
        }
    }

    return rectangles;
}

/**
 * Finds the
 * @param {n
 * @param {boolean[][]} visited - Matrix tracking visited cells
```

```
Support for loading ES Module in require() is an experimental feature and might change at
any time
(Use `node --trace-warnings ...` to show where the warning was created)

added 181 packages, and audited 182 packages in 12s

29 packages are looking for funding
  run `npm fund` for details

3 high severity vulnerabilities

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.
sergiopereira@Sergios-MBP demoAppWindsurf %
```

Great! I've created a complete Kanban board application with the following features:

1. Frontend:
   - Modern UI using Chakra UI
   - Drag and drop functionality using react-beautiful-dnd
   - Modal for creating new tasks
   - Real-time task updates
2. Backend:
   - Express server with MongoDB integration
   - OpenAI API integration for generating task instructions
   - RESTful API endpoints for task management

To run the application:

1. First, make sure MongoDB is running locally
2. Update the `.env` file with your OpenAI API key
3. Open two terminal windows and run:
   - Terminal 1 (backend): `npm run dev` from the project root
   - Terminal 2 (frontend): `cd client && npm start`

The application will be available at:

- Frontend: http://localhost:3000
- Backend API: http://localhost:5000

Note: You'll need to provide your OpenAI API key in the `.env` file before the application will work properly.

Would you like me to start the application for you?

Figure 1-5. Windsurf UI

Tool Comparison

I evaluated more than 30 AI tools in order to shortlist the ones I highlight in this chapter. Every tool covered here meets the following criteria:

- It is a professional project with a competent team behind it.
- The code it generates has a high quality threshold.
- It offers some level of functionality for free or on a trial basis.
- It had a high level of adoption at the time of writing (mid-2025).

My process in this chapter was as follows: I submitted a brief code challenge to each of the selected code tools, ran the same challenge exactly once in each tool, and compared their output. I then gave each tool a rating on a scale from 1 to 10, with 1 being the worst (a solution that errors out and doesn’t run at all) and 10 being a flawless solution. A 5 would be a solution that runs but solves only part of the problem.

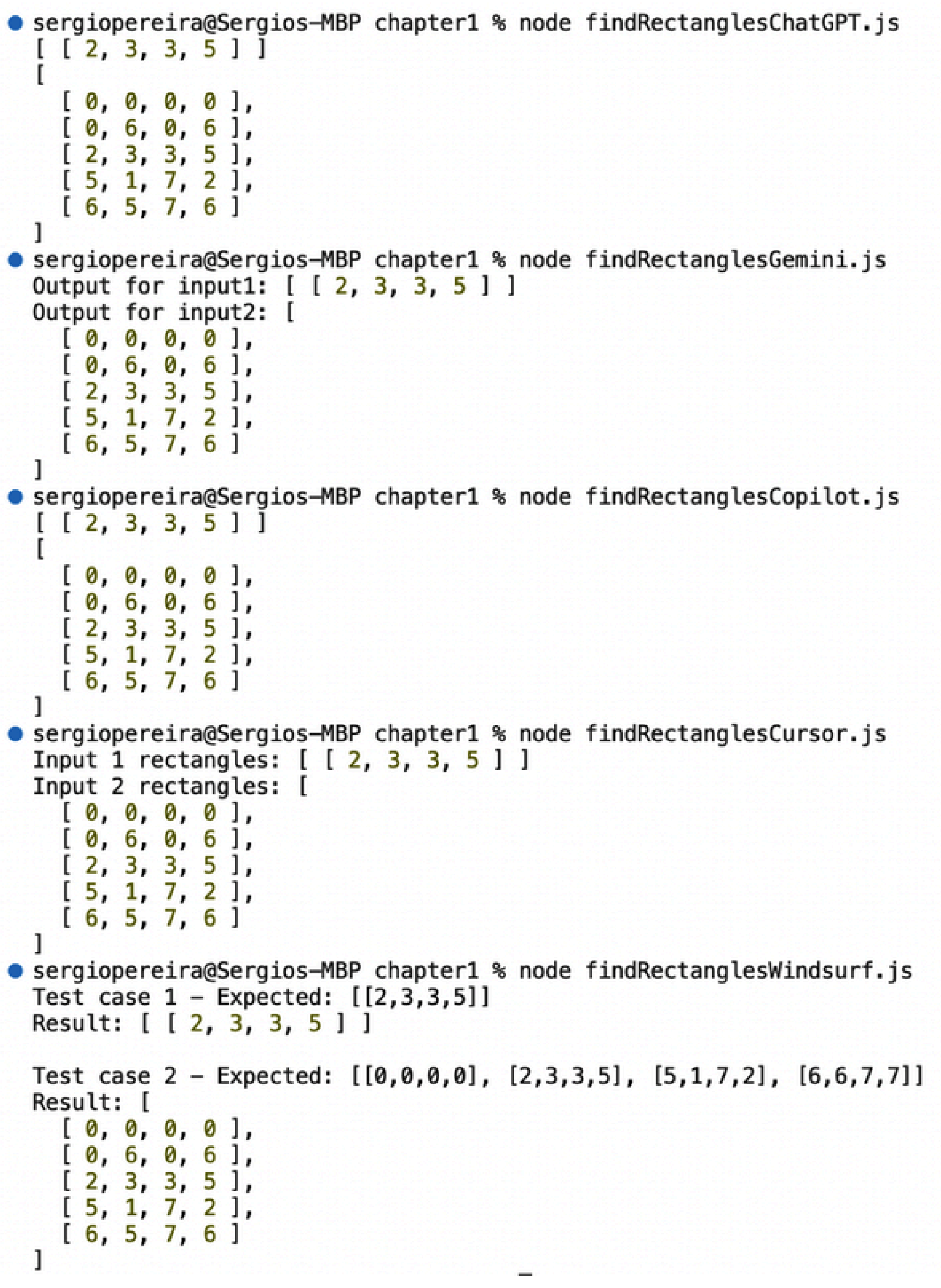
All tests described in this chapter were run in April 2025. Given the fast pace of evolution of each of these tools and their underlying models, it's likely that you could get a different result at a later time for the same prompt.

I gave each of the tools the same prompt, with a code challenge I've used as an interviewer in dozens of coding interviews:

```
Generate code in JavaScript to solve the following chal

Context:
- We have one 2D array, filled with zeros and ones.
- We have to find the starting point and ending point c
filled with 0.
- It is given that rectangles are separated and do not
however, they can touch the boundary of the array.
- A rectangle might contain only one element.

Desired output:
- You should return an array, each element representing
- Each of those array elements contains an array with 4
compose the rectangle (top left Y, top left X, bottom r

Example arrays:
input1 = [ [1, 1, 1, 1, 1, 1, 1],
           [1, 1, 1, 1, 1, 1, 1],
           [1, 1, 1, 0, 0, 0, 1],
           [1, 1, 1, 0, 0, 0, 1],
           [1, 1, 1, 1, 1, 1, 1],
           [1, 1, 1, 1, 1, 1, 1],
           [1, 1, 1, 1, 1, 1, 1],
           [1, 1, 1, 1, 1, 1, 1] ]

input2 = [ [0, 1, 1, 1, 1, 1, 0],
           [1, 1, 1, 1, 1, 1, 1],
           [1, 1, 1, 0, 0, 0, 1],
           [1, 1, 1, 0, 0, 0, 1],
           [1, 1, 1, 1, 1, 1, 1],
           [1, 0, 0, 1, 1, 1, 1],
           [1, 0, 0, 1, 1, 0, 0],
           [1, 0, 0, 1, 1, 0, 0] ]
```

This is a 2D array challenge, an algorithmic puzzle. Usually, I would start an hour-long live-coding interview by giving the candidate the challenge brief pretty much exactly as I've given it here to each of the tools. The candidates then code the solution, thinking out loud as they work, occasionally searching Google for help.

In that hour-long interview, very few candidates have ever managed to solve the full scope of the challenge (multiple rectangles). Most candidates would produce partial solutions that find only one rectangle, or only the top left corners, or some other variation. Figure 1-6 shows a screenshot I took of the console log after running the solutions tool.



```
● sergiopereira@Sergios-MBP chapter1 % node findRectanglesChatGPT.js
  [ [ 2, 3, 3, 5 ] ]
  [
    [ 0, 0, 0, 0 ],
    [ 0, 6, 0, 6 ],
    [ 2, 3, 3, 5 ],
    [ 5, 1, 7, 2 ],
    [ 6, 5, 7, 6 ]
  ]
● sergiopereira@Sergios-MBP chapter1 % node findRectanglesGemini.js
  Output for input1: [ [ 2, 3, 3, 5 ] ]
  Output for input2: [
    [ 0, 0, 0, 0 ],
    [ 0, 6, 0, 6 ],
    [ 2, 3, 3, 5 ],
    [ 5, 1, 7, 2 ],
    [ 6, 5, 7, 6 ]
  ]
● sergiopereira@Sergios-MBP chapter1 % node findRectanglesCopilot.js
  [ [ 2, 3, 3, 5 ] ]
  [
    [ 0, 0, 0, 0 ],
    [ 0, 6, 0, 6 ],
    [ 2, 3, 3, 5 ],
    [ 5, 1, 7, 2 ],
    [ 6, 5, 7, 6 ]
  ]
● sergiopereira@Sergios-MBP chapter1 % node findRectanglesCursor.js
  Input 1 rectangles: [ [ 2, 3, 3, 5 ] ]
  Input 2 rectangles: [
    [ 0, 0, 0, 0 ],
    [ 0, 6, 0, 6 ],
    [ 2, 3, 3, 5 ],
    [ 5, 1, 7, 2 ],
    [ 6, 5, 7, 6 ]
  ]
● sergiopereira@Sergios-MBP chapter1 % node findRectanglesWindsurf.js
  Test case 1 - Expected: [[2,3,3,5]]
  Result: [ [ 2, 3, 3, 5 ] ]

  Test case 2 - Expected: [[0,0,0,0], [2,3,3,5], [5,1,7,2], [6,6,7,7]]
  Result: [
    [ 0, 0, 0, 0 ],
    [ 0, 6, 0, 6 ],
    [ 2, 3, 3, 5 ],
    [ 5, 1, 7, 2 ],
    [ 6, 5, 7, 6 ]
  ]
```

Figure 1-6. Results from executing each tool's solution for the code challenge

As seen in Figure 1-6, all five tools returned correct results for the code challenge. The code for each of them is available on the book's GitHub repository. ChatGPT, Gemini, and Windsurf all generated solutions with clear and efficient code; I have nothing negative to say about their results. Copilot and Cursor used a depth-first search algorithm, which is typically used for traversing trees and is overkill for this specific problem. This added complexity would allow these tools to work for shapes other than rectangles, which was not a requirement here.

In any case, regardless of my code analysis, all these tools took a few seconds to produce perfect results for a code challenge that very few candidates can solve in an hour.

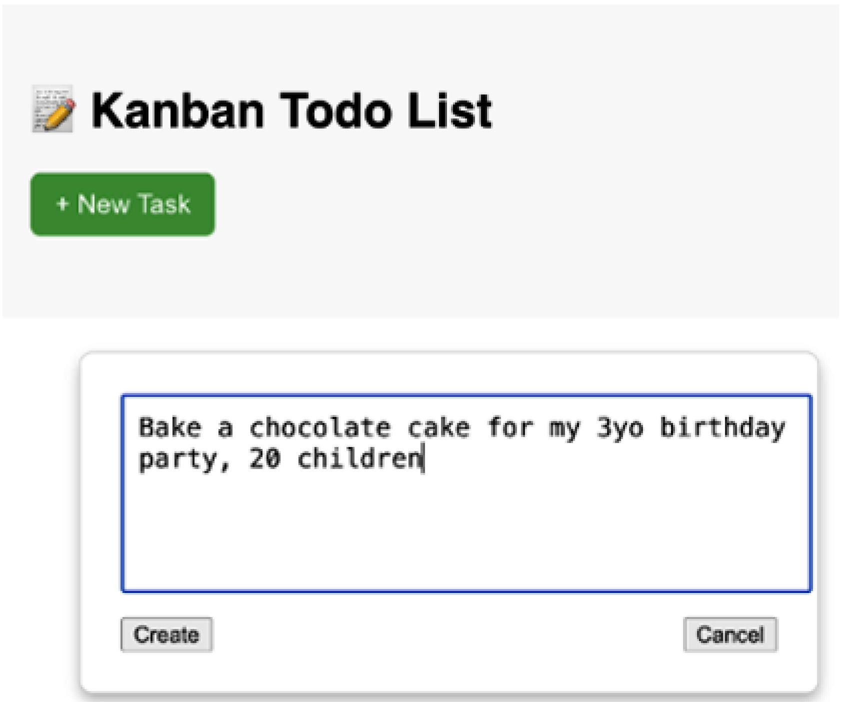
This shows me that this type of code challenge is just too simple for the current state of development of these AI tools. I'd rate all of them 10/10 based on this test. As such, I decided to give each tool a second, more complex challenge, asking them to create a full working application:

```
I want to build a to-do list app with these requirement
- The UI should be simple, with a Kanban board and a "N
- That "New task" button opens a modal to create a new
simple textarea field
- When a new task is added, the backend automatically p
instructions using an API integration with OpenAI
- All notes are stored in a database, each note has thr
user-created task title, the instructions pulled from O
a timestamp
```

Again, you can see the solutions produced by each of the five tools in this book's GitHub repository. Next are screenshots and my analysis of each tool's solution.

ChatGPT

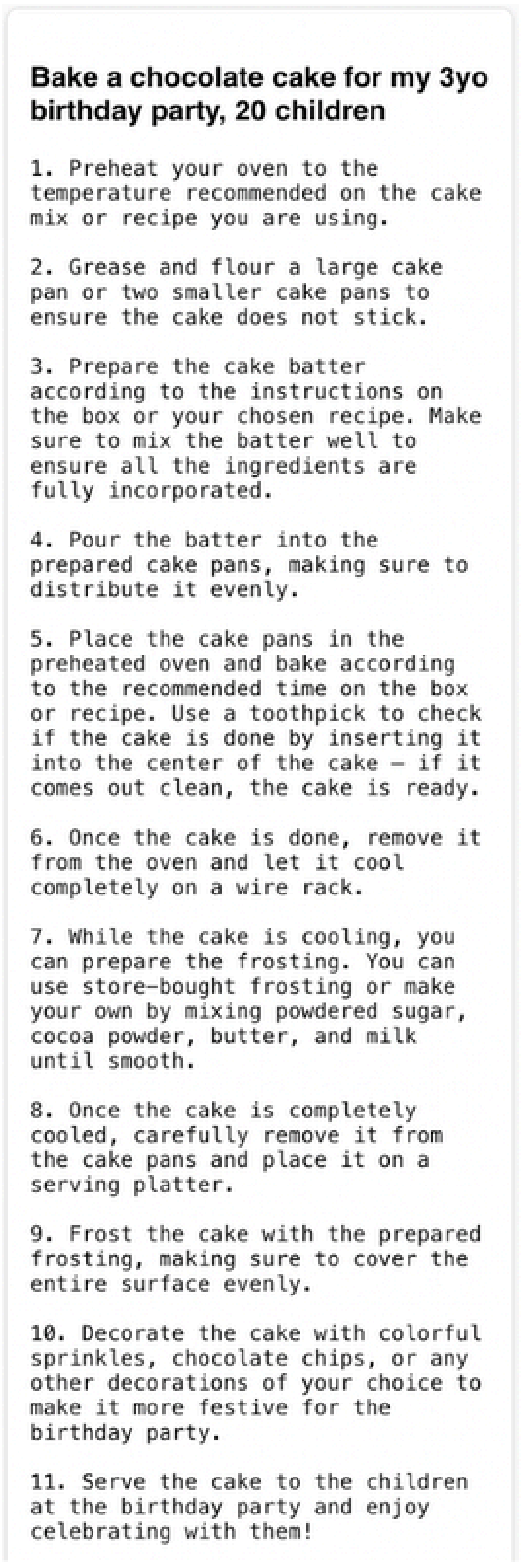
ChatGPT's solution worked well. I found it easy to understand the repository structure and to copy/paste all contents into the right files, and it was proficient walking me through some config errors I got when I first ran the solution. Within a few minutes I had the app working on my browser, as seen in Figure 1-7.



```
Bake a chocolate cake for my 3yo birthday
party, 20 children
```

Figure 1-7. App created in ChatGPT

The default state has the “New task” button, and once I click that button, a modal pop-up where I can write my task appears in the center of the screen. Once I click the Create button, it takes a few seconds to make the request to the OpenAI API on the backend. Finally, it displays the card with the instructions for the task, as generated by ChatGPT (Figure 1-8).


Bake a chocolate cake for my 3yo birthday party, 20 children

```
1. Preheat your oven to the
temperature recommended on the cake
mix or recipe you are using.

2. Grease and flour a large cake
pan or two smaller cake pans to
ensure the cake does not stick.

3. Prepare the cake batter
according to the instructions on
the box or your chosen recipe. Make
sure to mix the batter well to
ensure all the ingredients are
fully incorporated.

4. Pour the batter into the
prepared cake pans, making sure to
distribute it evenly.

5. Place the cake pans in the
preheated oven and bake according
to the recommended time on the box
or recipe. Use a toothpick to check
if the cake is done by inserting it
into the center of the cake - if it
comes out clean, the cake is ready.

6. Once the cake is done, remove it
from the oven and let it cool
completely on a wire rack.

7. While the cake is cooling, you
can prepare the frosting. You can
use store-bought frosting or make
your own by mixing powdered sugar,
cocoa powder, butter, and milk
until smooth.

8. Once the cake is completely
cooled, carefully remove it from
the cake pans and place it on a
serving platter.

9. Frost the cake with the prepared
frosting, making sure to cover the
entire surface evenly.

10. Decorate the cake with colorful
sprinkles, chocolate chips, or any
other decorations of your choice to
make it more festive for the
birthday party.

11. Serve the cake to the children
at the birthday party and enjoy
celebrating with them!
```

Figure 1-8. Instructions generated by ChatGPT

In my opinion, ChatGPT performed very well in this task, in terms of the code it generated, the instructions to run the code, and the debugging I requested. I still find it a bit burdensome that with ChatGPT, as with any other browser-based assistant, I have to copy/paste the contents of all files into my IDE. Even such a basic project as this to-do list app contains over a dozen files, nested inside client and server folders and subfolders under those. Even at this level, it gets error-prone and hard to track changes, which is a big barrier to using ChatGPT and other browser-based assistants on complex projects with more extensive codebases.

From a code-review perspective, ChatGPT's solution is a classic React.js and Express.js skeleton that fits the brief cleanly with a Kanban UI, "new task" modal, and working OpenAI call, all tucked behind environment variables. The codebase is structured in an intuitive client/server split and uses modern React hooks, async/await, and modest error handling.

On the downside, every task lives in RAM, so a server reboot wipes the board, and there's zero input validation, auth, or rate-limiting. Those omissions keep it firmly in "prototype" territory, but the absence of any outright security vulnerabilities makes it a safe starting point from which to harden and extend. ChatGPT's browser-based nature made it challenging to copy/paste the code into the right files. But after that, the solution ran without too many lengthy iterations.

The solution has a basic UI but it does fulfill the brief, and the code quality is quite good. With all this, I rate ChatGPT an 8/10 for this test.

Google Gemini

Gemini's solution didn't work. I stuck with it for as long as I did for the other tools, and I got a working solution, as you can see from the code in the book's GitHub repository. Both the frontend and backend run, and I got rid of some errors I got along the way, which Gemini helped me fix.

However, the best Gemini gave me was a blank window, as seen in Figure 1-9.

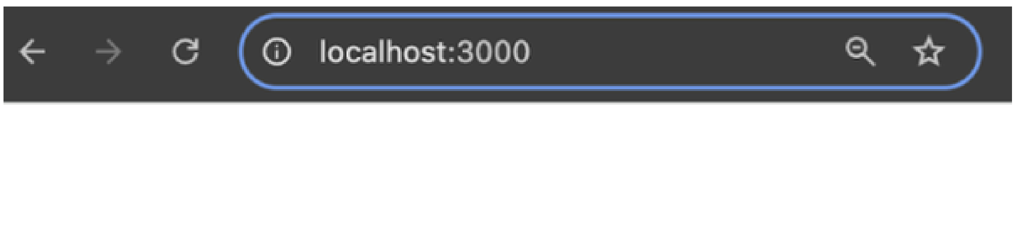
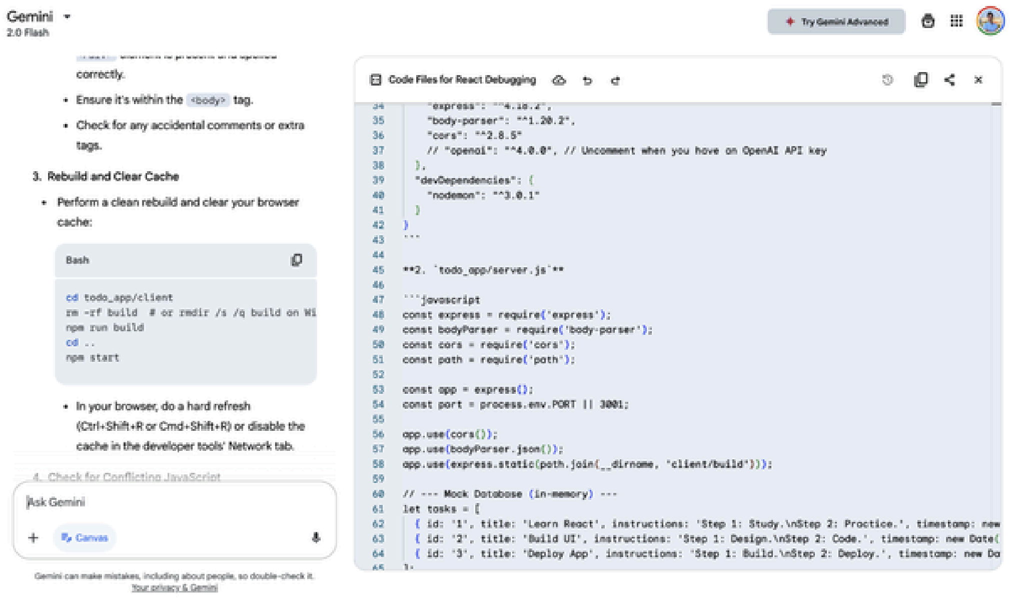


Figure 1-9. Gemini generated a blank window

Besides the fact that the solution didn't work, Gemini makes it really slow and error-prone to generate code that is more complex than a single file. Unlike ChatGPT, Gemini generates all code in a single big file in its code editor (see the right panel in Figure 1-10). This makes it very hard to copy the code into my IDE, since I need to manually create each folder and file and copy the contents there. It also makes it very difficult to track changes made during debugging.



correctly.

- Ensure it's within the `<body>` tag.
- Check for any accidental comments or extra tags.

3. Rebuild and Clear Cache

- Perform a clean rebuild and clear your browser cache:

```bash
cd todo_app/client
rm -rf build  # or rmdir /s /q build on Wi
npm run build
cd ..
npm start
```

- In your browser, do a hard refresh (Ctrl+Shift+R or Cmd+Shift+R) or disable the cache in the developer tools' Network tab.

4. Check for Conflicting JavaScript

Gemini can make mistakes, including about people, so double-check it. Your privacy & Gemini

~~~
    "express": "^4.18.2",
    "body-parser": "^1.20.2",
    "cors": "^2.8.5"
    // "openai": "^4.0.0", // Uncomment when you have an OpenAI API key
  },
  "devDependencies": {
    "nodemon": "^3.0.1"
  }
}
```

**2. `todo_app/server.js`**

```javascript
const express = require('express');
const bodyParser = require('body-parser');
const cors = require('cors');
const path = require('path');

const app = express();
const port = process.env.PORT || 3001;

app.use(cors());
app.use(bodyParser.json());
app.use(express.static(path.join(__dirname, 'client/build')));

// --- Mock Database (in-memory) ---
let tasks = [
  { id: '1', title: 'Learn React', instructions: 'Step 1: Study.\nStep 2: Practice.', timestamp: new
  { id: '2', title: 'Build UI', instructions: 'Step 1: Design.\nStep 2: Code.', timestamp: new Date(
  { id: '3', title: 'Deploy App', instructions: 'Step 1: Build.\nStep 2: Deploy.', timestamp: new Da
];
~~~

Figure 1-10. Gemini UI during the test

For all these reasons, Gemini's performance in this second test was very disappointing.

When I reviewed Gemini's code, I realized that it aimed for a stylish frontend that could have resulted in a crisp and modern UI, with TypeScript, Tailwind, Framer Motion, and Hero icons—except it didn't work. There was also an obvious error in the React code, where it was exporting the app component but not actually rendering it. This feels like an error that Gemini should have easily caught.

On the backend, the code seems to contain the desired functionality. However, it exposes the OpenAI API key in the actual code rather than using an environment variable. That's a serious security issue that would make this solution unacceptable for deployment or even testing.

Gemini provided the most disappointing experience of all the tools reviewed. First there was the clunky and error-prone experience of copy/pasting all code into the right files, then the errors in running both the frontend and backend, and then spinning its wheels to fix the React issues that prevented the UI from rendering properly. I ended up with no working solution even after spending significant time on this. With the caveat that Gemini produced a working solution for the first challenge and some relevant code for this second challenge, I rate it a 4/10.

GitHub Copilot

Copilot's solution ran nicely at first, since it uses minimal libraries and dependencies. It required a few debugging iterations when dealing with the structure of the OpenAI API request: first it was using the wrong endpoint, then the payload was structured wrong, then it had some parsing errors. But after a few minutes I had the working solution shown in Figures 1-11 and 1-12, with a very clunky frontend composed of vanilla HTML without much styling.

The button works, and when I add a new task, it does trigger the backend functionality. However, the frontend doesn't hide the modal pop-up when I'm done, nor does it have any "x" button to close it. It's visually underwhelming, and improving the UI would take multiple iterations.

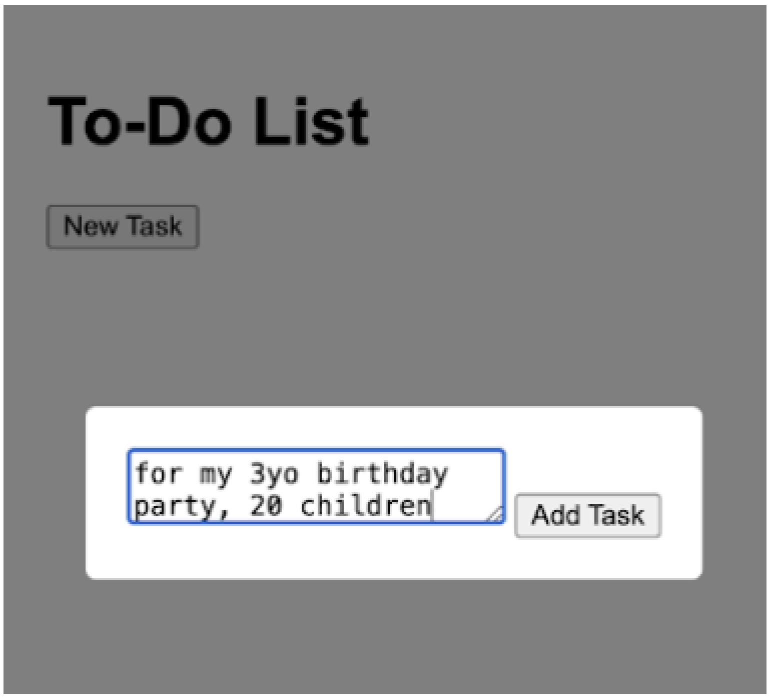
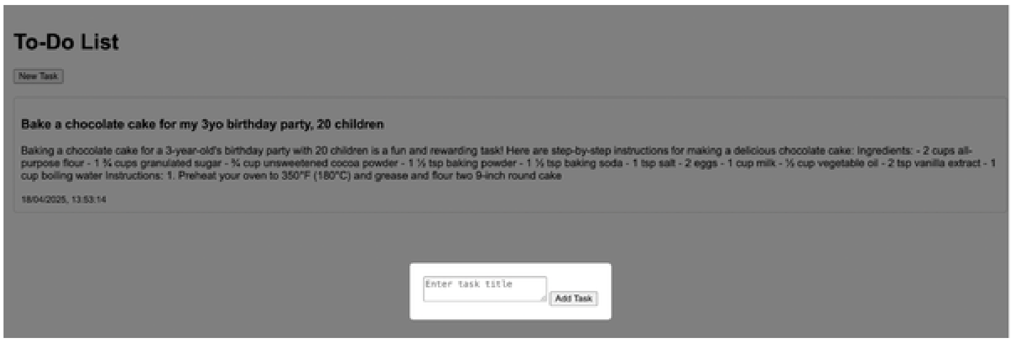


Figure 1-11. GitHub Copilot's solution—modal pop-up



Bake a chocolate cake for my 3yo birthday party, 20 children

Baking a chocolate cake for a 3-year-old's birthday party with 20 children is a fun and rewarding task! Here are step-by-step instructions for making a delicious chocolate cake: Ingredients: - 2 cups all-purpose flour - 1 ¾ cups granulated sugar - ¾ cup unsweetened cocoa powder - 1 ½ tsp baking powder - 1 ½ tsp baking soda - 1 tsp salt - 2 eggs - 1 cup milk - ½ cup vegetable oil - 2 tsp vanilla extract - 1 cup boiling water Instructions: 1. Preheat your oven to 350°F (180°C) and grease and flour two 9-inch round cake

18/04/2025, 13:53:14

Figure 1-12. GitHub Copilot's solution—task added

From a code-review perspective, Copilot created a minimalist proof-of-concept, with plain HTML, CSS, and vanilla JavaScript on the frontend; a tiny

Express server on the backend; and hardly any dependencies. The upside is total simplicity: anyone could read this code in minutes, and deployments are lightweight.

The trade-off is that every best-practice box stays unchecked: the OpenAI key is hardcoded. There's no database, no validation, no responsive design, and no framework ergonomics for future growth. It would be a great solution for a hackathon, but risky for anything user-facing without a major refactor.

I think there's merit in this approach, given that I didn't prompt Copilot with any specifics about the code quality or robustness I wanted. As a CTO, I appreciate starting simply, and Copilot did produce the simplest app of all tools reviewed for this challenge. I rate it 8/10 in this comparison test.

Cursor

Cursor separated its code into frontend and backend folders, and it was very easy to run each of them in Cursor with simple NPM start commands. The backend flow worked nicely. Both the database and the integration with the OpenAI API worked well, too, but the frontend was clunky. The baseline UI is weird, with the title on the very top and the "New task" button on the very bottom, as seen in Figure 1-13.

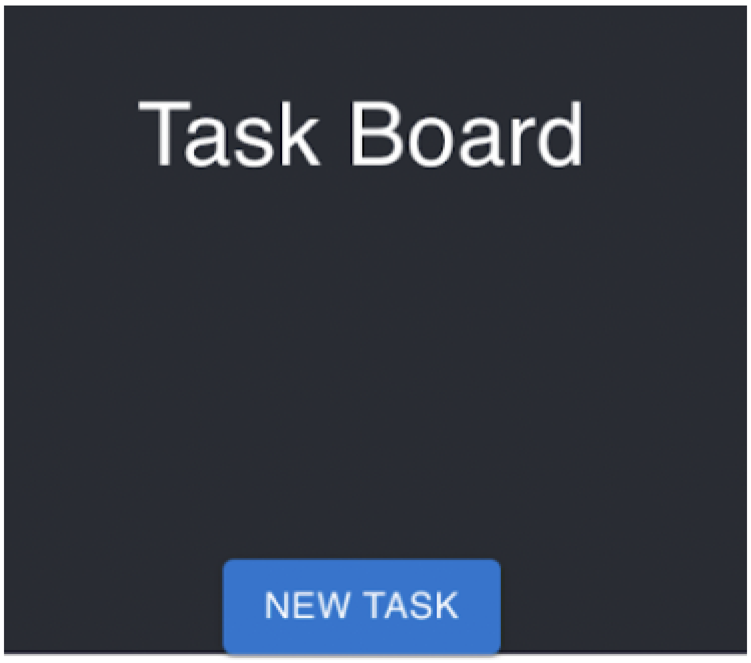
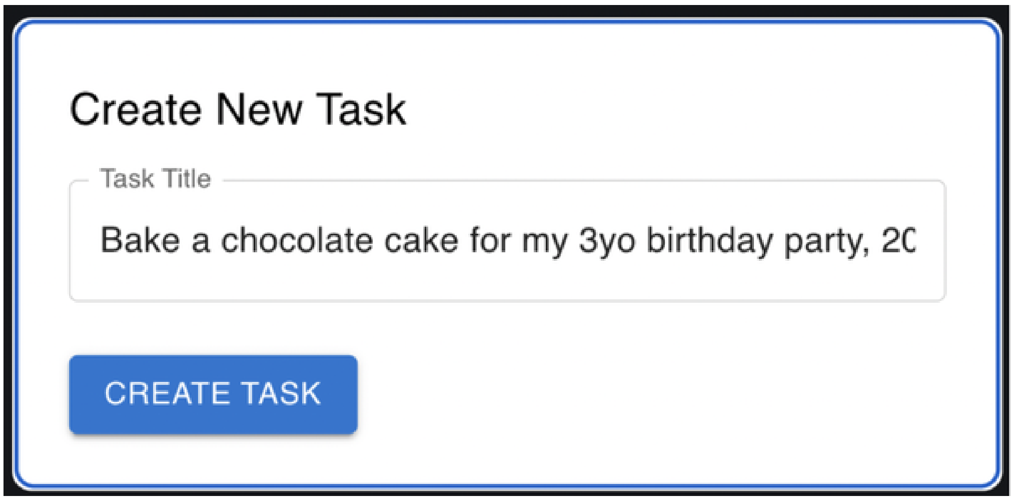


Figure 1-13. Cursor's solution UI

Once I click the "New task" button, a pop-up appears. It has decent styling but a one-line input field that cuts off any longer note descriptions, as seen in Figure 1-14.

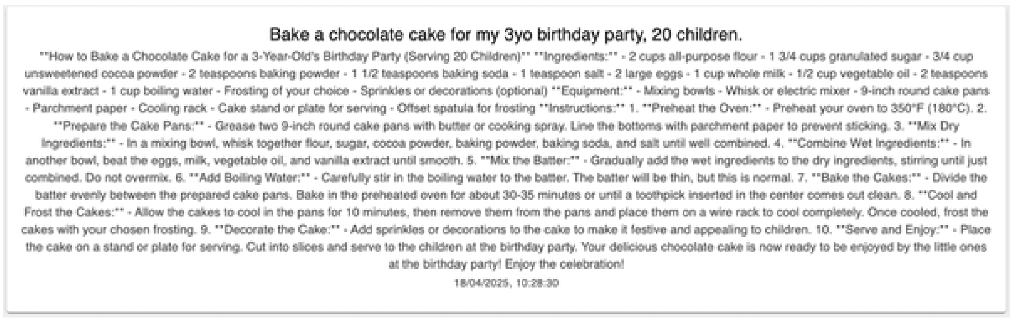


Task Title

Bake a chocolate cake for my 3yo birthday party, 2C

Figure 1-14. Cursor’s solution UI pop-up

Once a task is created in the modal pop-up, the task cards are rendered below the button, at the very bottom of the screen (Figure 1-15).



Bake a chocolate cake for my 3yo birthday party, 20 children.

How to Bake a Chocolate Cake for a 3-Year-Old's Birthday Party (Serving 20 Children) **Ingredients:** - 2 cups all-purpose flour - 1 3/4 cups granulated sugar - 3/4 cup unsweetened cocoa powder - 2 teaspoons baking powder - 1 1/2 teaspoons baking soda - 1 teaspoon salt - 2 large eggs - 1 cup whole milk - 1/2 cup vegetable oil - 2 teaspoons vanilla extract - 1 cup boiling water - Frosting of your choice - Sprinkles or decorations (optional) **Equipment:** - Mixing bowls - Whisk or electric mixer - 9-inch round cake pans - Parchment paper - Cooling rack - Cake stand or plate for serving - Offset spatula for frosting **Instructions:** 1. **Preheat the Oven:** - Preheat your oven to 350°F (180°C). 2. **Prepare the Cake Pans:** - Grease two 9-inch round cake pans with butter or cooking spray. Line the bottoms with parchment paper to prevent sticking. 3. **Mix Dry Ingredients:** - In a mixing bowl, whisk together flour, sugar, cocoa powder, baking powder, baking soda, and salt until well combined. 4. **Combine Wet Ingredients:** - In another bowl, beat the eggs, milk, vegetable oil, and vanilla extract until smooth. 5. **Mix the Batter:** - Gradually add the wet ingredients to the dry ingredients, stirring until just combined. Do not overmix. 6. **Add Boiling Water:** - Carefully stir in the boiling water to the batter. The batter will be thin, but this is normal. 7. **Bake the Cakes:** - Divide the batter evenly between the prepared cake pans. Bake in the preheated oven for about 30-35 minutes or until a toothpick inserted in the center comes out clean. 8. **Cool and Frost the Cakes:** - Allow the cakes to cool in the pans for 10 minutes, then remove them from the pans and place them on a wire rack to cool completely. Once cooled, frost the cakes with your chosen frosting. 9. **Decorate the Cake:** - Add sprinkles or decorations to the cake to make it festive and appealing to children. 10. **Serve and Enjoy:** - Place the cake on a stand or plate for serving. Cut into slices and serve to the children at the birthday party. Your delicious chocolate cake is now ready to be enjoyed by the little ones at the birthday party! Enjoy the celebration!

18/04/2025, 10:28:30

Figure 1-15. Pop-up in Cursor’s solution

Cursor’s solution actually implemented its functionality on the first try, making it very easy for me to run both frontend and backend. However, this solution didn’t look like the Kanban board I asked for. It’s a little better than Copilot’s UI, but still a bit basic.

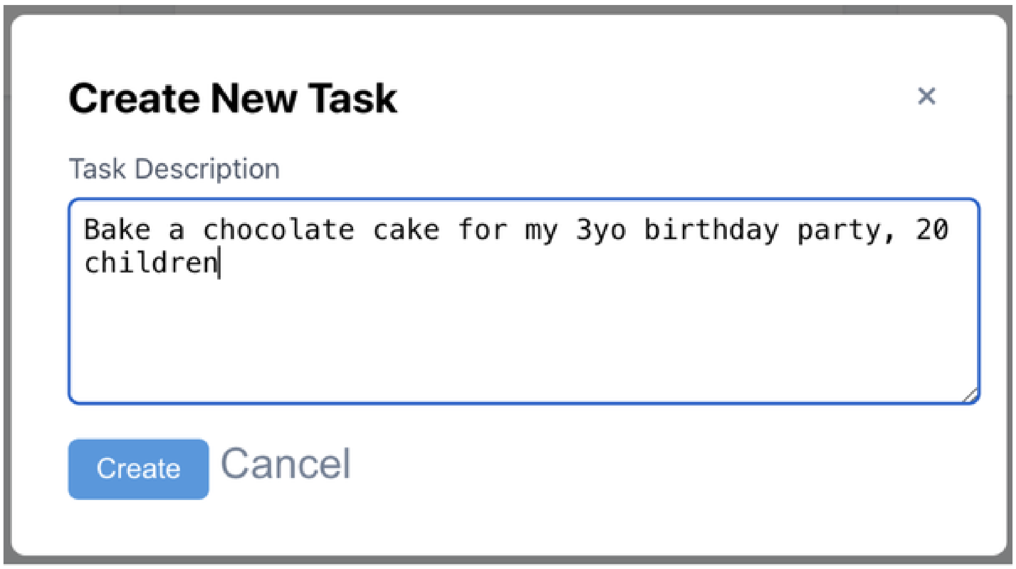
Looking into the code, I can tell that Cursor aimed for a sleek UI, using Material UI and react-beautiful-dnd. For some reason, though, that didn’t translate into a pleasant UI. It also has the code well componentized and correctly uses environment variables, so the basics are solid. The backend is a bit barebones, with everything inside a single Express file with an in-memory array. There is no input validation and no error handling.

In a nutshell, Cursor’s developer experience was unbeatable: it produced working code, and both the frontend and backend ran on the first try. The code works and fulfills the functionality, even if the UI isn’t great and there’s plenty of room for improvement under the hood. I rate Cursor a 9/10.

Windsurf

I found it challenging to get Windsurf’s solution to work. It initially proposed an overly complex solution with unnecessary dependencies.

First, the frontend didn’t run at all. Windsurf went down a “vibe debugging” rabbit hole with broken dependencies from Chakra UI, a frontend library that it decided to use. Eventually, I had to prompt it to not use that library at all so we could move on. Once the library was removed, the frontend finally ran (Figure 1-16).



Create New Task

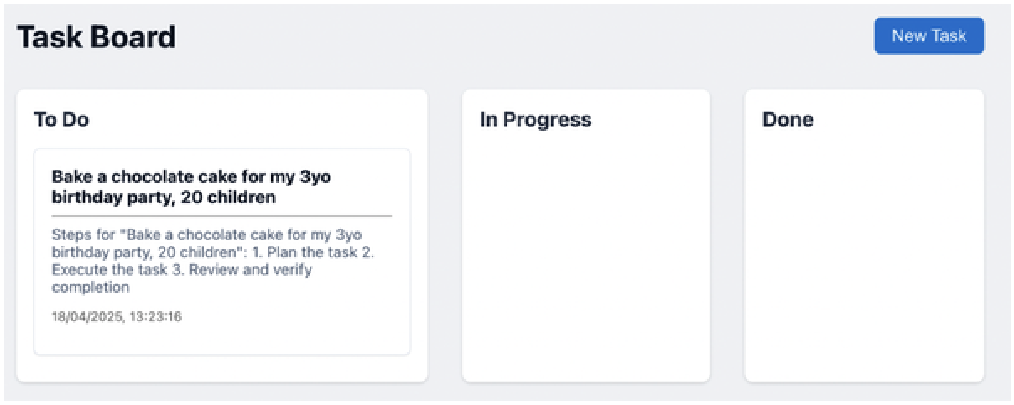
Task Description

Bake a chocolate cake for my 3yo birthday party, 20 children

Figure 1-16. Windsurf solution UI

Then I had a similar challenge on the backend, this time with MongoDB dependencies. For some reason, Windsurf kept going down rabbit holes with such a simple task, and I ultimately had to prompt it to simplify the backend and use in-memory storage.

It finally ran and placed the task I created in the To Do column, in a proper Kanban UI (Figure 1-17).



Task Board

To Do

Bake a chocolate cake for my 3yo birthday party, 20 children

Steps for "Bake a chocolate cake for my 3yo birthday party, 20 children": 1. Plan the task 2. Execute the task 3. Review and verify completion

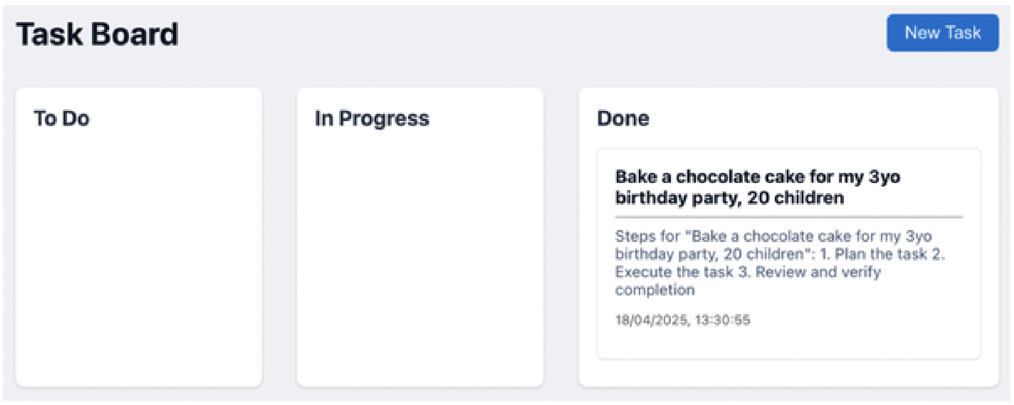
18/04/2025, 13:23:16

In Progress

Done

Figure 1-17. Windsurf Kanban board

However, it required me to drag the task to the In Progress column in order to trigger the OpenAI API request—but then crashed when I did so (Figure 1-18).



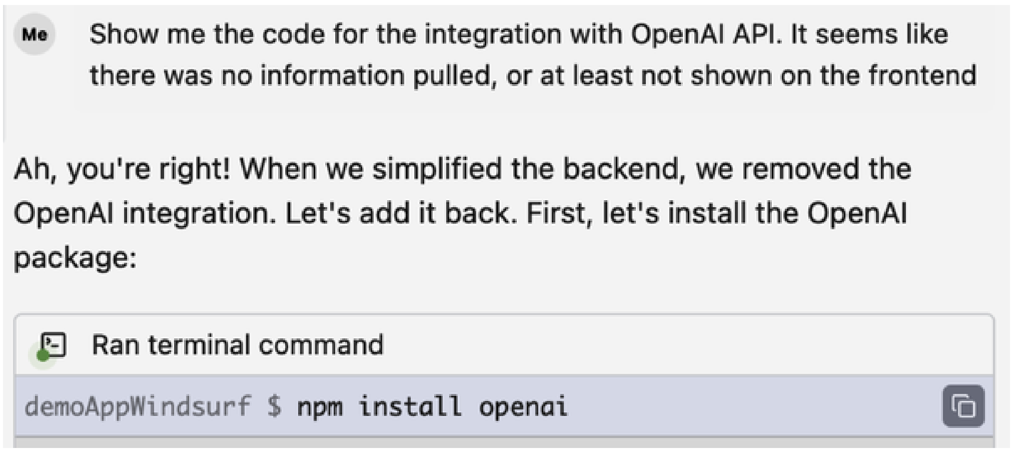
Bake a chocolate cake for my 3yo birthday party, 20 children

Steps for "Bake a chocolate cake for my 3yo birthday party, 20 children": 1. Plan the task 2. Execute the task 3. Review and verify completion

18/04/2025, 13:30:55

Figure 1-18. Windsurf solution

Again, I gave Windsurf more time to debug the issue, and finally, the app ran without errors. However, I found that it ran a bit "too fast," and the result was underwhelming. As it turned out, Windsurf had accidentally removed the code for the OpenAI API integration, one of the key features of the app (Figure 1-19).



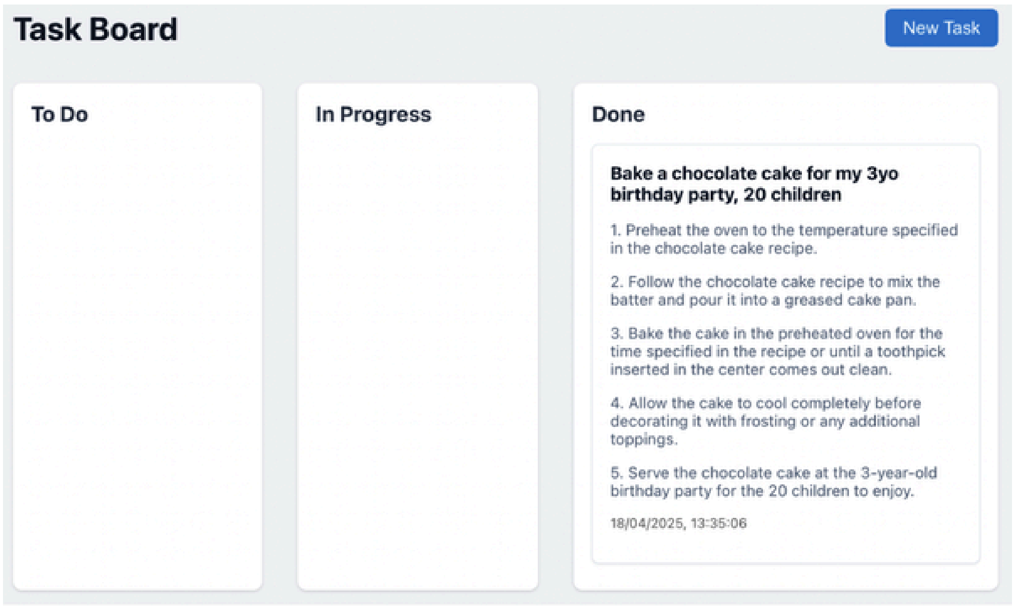
Show me the code for the integration with OpenAI API. It seems like there was no information pulled, or at least not shown on the frontend

Ah, you're right! When we simplified the backend, we removed the OpenAI integration. Let's add it back. First, let's install the OpenAI package:

```
demoAppWindsurf $ npm install openai
```

Figure 1-19. Windsurf accidentally removed the OpenAI integration

Finally, after a significant amount of time spent in these back-and-forth iterations, I finally had a working Windsurf solution that has an elegant Kanban UI and fulfills the backend functionality (Figure 1-20).



Bake a chocolate cake for my 3yo birthday party, 20 children

1. Preheat the oven to the temperature specified in the chocolate cake recipe.
2. Follow the chocolate cake recipe to mix the batter and pour it into a greased cake pan.
3. Bake the cake in the preheated oven for the time specified in the recipe or until a toothpick inserted in the center comes out clean.
4. Allow the cake to cool completely before decorating it with frosting or any additional toppings.
5. Serve the chocolate cake at the 3-year-old birthday party for the 20 children to enjoy.

18/04/2025, 13:35:06

Figure 1-20. Finished Windsurf solution

In terms of code, Windsurf's solution is the most feature-complete: it uses React with drag-and-drop, modal creation, real-time refresh, and clean separation between routes, controllers, and components. Environment variables are respected, error handling is a bit more thoughtful, and the code style is consistent and easy to follow.

Interestingly, Windsurf took the most maximalist approach of this test. It tried to use fancy libraries that led it down rabbit holes it couldn't get out of on its own. I had to prompt it out of those three times, by instructing Windsurf to not use the dependencies that were creating trouble (first the Chakra UI library, then MongoDB, then the drag-and-drop). Its agentic approach makes Windsurf the most capable and seemingly independent tool I tested, even compared with Cursor and Copilot. However, that independence led it down dead ends a few times, and it takes an experienced developer to instruct it out of those dead ends.

Windsurf was also the tool that took me the longest to get to a working solution, over one hour. But it created the sleekest UI, and the code quality was quite good. I rate it a 9/10.

Table 1-1 compares my ratings for how well each of these tools performed in my testing.

Table 1-1. Code-generation tools overview

Tool	UX	Test performance
ChatGPT	Browser	8/10
Google Gemini	Browser	4/10
GitHub Copilot	IDE	8/10
Cursor	IDE	9/10
Windsurf	IDE	9/10

Conclusion

I’ve used the first test, the 2D-array code challenge, dozens of times in interviews over the years. It’s incredible that all of the five tools reviewed in this test can produce the same outcomes as top-performing software engineers in only 10 seconds.

Clearly, these tools vastly outperform humans for these types of tasks: that is, tasks with clear requirements and input/output guidelines, where the code produced fits in a single file and for which there’s plenty of relevant training data. All these interview-type challenges fit such a description, and more broadly, any of these LLM tools could likely solve any such algorithmic puzzle easily within seconds.

When I gave them challenges that included producing more extensive code repositories, some tools began facing difficulties, even for a simple Kanban UI with notes and backend automation. Using browser-based tools, such as ChatGPT and Google Gemini, makes it very burdensome to copy/paste the code into the right files. It’s a lot of work, and it’s very error-prone. It’s an inferior experience compared to IDE-based tools that have both the code and the console terminal within their context window. As the developer, I don’t need to copy/paste anything; it’s all about reviewing and accepting or rejecting the suggestions. IDE-based tools make it feel like an efficient code review cycle, where I ask for changes, it codes them on the spot, and I review, accept, and test within seconds.

It’s also clear that multiple state-of-the-art models from different providers are already capable of producing good software code. All of the IDE-based tools I tested (Cursor, Windsurf, and GitHub Copilot) allow developers to select a model from a drop-down list. Browser-based tools offer a more closed environment, with ChatGPT using only OpenAI's models and Gemini using only Google's models.

This means that the competition in this market is increasingly about the actual developer experience. Browser-based tools are out of this race, in my opinion. It's just too burdensome to share context with them, and then even more burdensome to copy/paste the code back to the repo. The winning solutions in this space will be based in the IDE, not the browser.

Even within IDE-based tools, there are different approaches. GitHub Copilot still has the original vibe of bringing a ChatGPT-like chat into the IDE, with access to the code and the ability to make changes. On the other hand, Windsurf takes it to a higher level of abstraction, with an agentic approach that creates folders and files, makes changes, and gives me a simple button to review the changes and "accept all," just like in a code review. Cursor is somewhere in the middle, with a better chat UX than Copilot, and provides users with the option to use its agentic mode (which I didn't use for this test).

From a software engineer's perspective, it's clear that our future work will be less about writing actual code and more about prompting these tools and reviewing the code they generate. This sounds much easier than it actually is, though. These tools create significant potential for driving entire codebases into a buggy and unmaintainable state, just like my scenario with Windsurf. Imagine a production-level codebase with hundreds of files, and Windsurf using its agentic mode to add random libraries and remove key parts of the backend functionality. I can see the potential for these tools to do a lot of harm if operated on a "vibe coding" basis, with developers mindlessly accepting suggestions without properly reviewing and testing the code.

As a software engineer, you should absolutely use these tools: they can help you ship product features faster and, in many cases, with a higher quality standard. However, you should *always* review AI-generated code before pushing it to production or opening a pull request. Make the code yours, regardless of how much of it was generated by an AI tool. There's decent potential for these tools to wreck your working code, so you should run the code they generate against a test suite that covers a wide range of cases, from the happy path to edge cases and error states. Getting all tests green is a solid confirmation that the code fulfills your requirements. And finally, be sure to revisit your company’s guidelines for any AI tools you use for professional purposes.

1 Gülen, Kerem. April 15, 2025. “ChatGPT Just Hit 1 Billion Users and Melted Its Own Servers”. *Dataconomy*.

2 Palazzolo, Stephanie, and Amir Efrati. April 1, 2025. “ChatGPT Revenue Surges 30%—in Just Three Months”. *The Information*.

3 Sentisight.ai. January 24, 2025. “Google Gemini: How Has It Been Received by Users so Far?”

4 Millward, Wade Tyler. July 31, 2024. “Microsoft Q4 2024: CEO Nadella Calls Copilot Growth Rate Fastest for Any M365 Suite”. *CRN*.

5 GitHub Copilot Resources. “Measuring the Impact of GitHub Copilot”. GitHub, accessed June 4, 2025.

6 Shibu, Sherin. April 9, 2025. “This AI Startup Spent $0 on Marketing. Its Revenue Just Hit $200 Million in March”. *Entrepreneur*.