TL;DR Quick Start

Step	Description	Prompt / Script
1	Scan Your Code	Open files/modules in Cursor IDE. Use targeted prompts to identify error-prone areas (unhandled promises, null access, deprecated APIs).
2	Generate Fix Suggestions	Highlight problematic code and prompt: Cursor, suggest a fix for [specific bug] in [file]. Review multiple options and select the best.
3	Validate Fixes	Run unit and integration tests on modified code. Include edge cases Cursor may not cover.
4	Apply Fix and Commit	Implement chosen fix and use clear commit messages describing the issue and solution.
5	Monitor Recurring Bugs	Track recurring issues in logs/issue tracker. Use Cursor to search for similar patterns and adjust prompts/templates for future bugs.

Setting Up Cursor for Debugging

Step 1 – Installation

- Download and install Cursor for your OS.
- Open your project/repository inside Cursor IDE (File → Open Folder).
- Optional: Install Cursor CLI/Agent to trigger analysis from terminal or CI/CD.

Step 2 – Initial Configuration

Task	Details
Language & Framework	Specify JS, TS, Python, React, Node.js, Django, etc.
Auto-suggestions	Enable live scanning and adjust frequency.
Alert Thresholds	Set severity: warnings, errors, critical issues.

Step-by-Step Bug Fixing Runbook

Step 1 – Identify Bugs

- Open project in Cursor IDE.

- Use targeted prompts to find error-prone code (unhandled promises, null access, deprecated APIs).
- Prioritize issues based on context and risk.

Step 2 – Generate Fix Suggestions

- Use Suggest Fix prompt in Cursor.
- Review multiple AI-generated solutions.
- Select the solution matching project architecture.

Example template:

```
try {
  const response = await fetchData();
  processResponse(response);
} catch (error) {
  logError(error);
  retryFetchData();
}
```

Step 3 – Validate Fixes

- Run unit and integration tests.
- Check edge cases and integration workflows.

Mini-Checklist: - Run relevant unit tests - Check integration workflows - Review Cursor's suggested test coverage

Step 4 – Apply Fix and Commit

- Implement fix in codebase.
- Use descriptive commit messages, e.g., `Fix: Handle async API rejection in userController.js`.
- Test in local/staging environment.

Step 5 – Monitor and Iterate

- Track recurring issues.
- Adjust prompts/templates for future bugs.

Mini-Checklist: - Scan & prioritize code - Review & apply fix suggestions - Run targeted tests - Commit with clear documentation - Track recurring issues

---

Advanced Cursor Debugging Tips

Tip #1: Prompt Engineering

- Craft prompts that guide Cursor to analyze code deeply.
- Include context from surrounding files or modules.
- Request multiple solutions for comparison.

Tip #2: Automating Repetitive Fixes

- Create workflow templates for recurring bugs.
- Enable conservative automation for multiple files.
- Review diffs and run tests to ensure safety.

Cursor Debugging Performance Metrics

Metric / Focus Area	Before Cursor	After Cursor	Insights / Actions
Avg Bug Resolution Time	3h	1.2h	Track fix time by bug type.
Recurring Bug Frequency	12/week	7/week	Use Cursor templates for recurring patterns.
Unit & Integration Test Failures	5%	1.5%	Validate fixes proactively.
High-Risk Code Modules	6 modules	2 modules	Focus scanning on frequent-error modules.
Developer Time Saved	N/A	2–3h/week	Allocate saved time to refactoring/ features.

Examples of Bug Fixes Using Cursor

Example 1 – Fixing API Timeout

```
async function fetchDataWithRetry(url, retries = 3) {
  for (let i = 0; i < retries; i++) {
    try {
      const response = await fetch(url);
      return await response.json();
```

```
    } catch (error) {
      if (i === retries - 1) throw error;
      await new Promise(res => setTimeout(res, 500 * (i + 1)));
    }
  }
}
```

Example 2 – Resolving Null Reference Errors

```
const userEmail = user?.profile?.email ?? "unknown@example.com";
```

Example 3 – Optimizing CSS Rendering Bugs

```
/* Original conflicting rules */
.button { color: red; }
.primary-button { color: blue; }

/* Optimized */
.button { color: red; }
.primary-button { color: blue !important; }
```

Common Challenges & Cursor Workarounds

Issue #1: Handling Missed Edge Cases

- Run multiple prompts with additional context.
- Add comprehensive test cases.
- Validate fixes manually.

Issue #2: Conflicting Suggestions

- Compare options carefully.
- Test each candidate fix.
- Choose the most compatible solution.
- Maintain a decision log.

Final Summary & Key Takeaways

- Follow the workflow step-by-step.

• Validate fixes with testing.
• Track patterns to prevent recurring bugs.
• Use advanced features: prompts, automation, test integrations.
• Investigate root causes for future prevention.

Cursor Bug Fixing Templates Pack

• 12 ready-to-use Cursor prompts for common issues.
• 5 step-by-step workflows.
• 4 checklists and metrics templates.
• 7 example code snippets.
• 3 workflow templates for multi-file or integration errors.
• Delivered in copy-paste-ready format.