Rapidly Build and Deploy a Full-Stack App with Cursor

Advanced AI-assisted SaaS application development

Lucas Soares

12/02/2025

Table of Contents

Cursor 2.0 & Context Management

AI-First Code Editor for Production Development

Not just autocomplete—full stack development

AI-First Code Editor for Production Development

Not just autocomplete—full stack development

⚡ 4x Faster

New Composer model completes tasks in <30s

🤖 8 Agents

Run parallel agents without conflicts

🔍 Semantic Search

Understand entire codebase contextually

🔗 MCP Integration

Connect to GitHub, Figma, databases

Core Features

💬 Chat (Cmd+L)

- Multi-tab conversations
- Context checkpoints
- Quick Q&A

⌨️ Tab Autocomplete

- Multi-line predictions
- Context-aware
- Pattern learning

🎯 Agent (Cmd+I)

- Multi-file editing
- Autonomous execution
- Planning + building

✏️ Inline Edit (Cmd+K)

- In-place modifications
- Natural language
- Stay in flow

The Three-Step Framework

1. EXPLORE

Share context
Identify files
Discuss options

2. PLAN

Step-by-step plan
Markdown checklists
Break down tasks

3. BUILD

Execute iteratively
Review each step
Commit regularly

The Three-Step Framework

1. EXPLORE

Share context
Identify files
Discuss options

2. PLAN

Step-by-step plan
Markdown checklists
Break down tasks

3. BUILD

Execute iteratively
Review each step
Commit regularly

💡 **This framework prevents context overload and ensures quality output**

[1] Cursor for Staff Engineers

Context Management: The Foundation

⚠ Key Challenge

Context management is like telling a story—when it gets convoluted, the AI loses track

Context Management: The Foundation

⚠ Key Challenge

Context management is like telling a story—when it gets convoluted, the AI loses track

Why It Matters

- Fixed context windows (128k-1M tokens)
- Comprehension ≠ token count
- Poor context = hallucinations

Best Practices

- Use @-mentions for precision
- Start new chats frequently
- Configure .cursorignore

The @ Symbol: Your Context Navigator

📄 @file

```
@src/components/Header.tsx
```

Reference specific files

📁 @folder

```
@src/utils/
```

Include entire directories

🔍 @codebase

```
@codebase "authentication"
```

Semantic search

📚 @docs

```
@docs "React hooks"
```

Query documentation

Effective Prompting

❌ Vague

```
Make this better
```

- No success criteria
- No direction
- Ambiguous goal

→

✅ Specific

```
@src/components/Button.tsx Refactor to: 1. Add TypeScript props 2. Include disabled state 3. Add loading spinner 4. Keyboard accessible
```

💡 **Describe the diff you want: current state → desired state**

Project Rules: .cursor/rules

Four Application Modes

1. **Always Apply** - Every session automatically
2. **Apply Intelligently** - AI determines relevance (recommended)
3. **Apply to Specific Files** - Pattern matching (*.test.ts)
4. **Apply Manually** - Activated with @-mention

Project Rules: .cursor/rules

Four Application Modes

1. **Always Apply** - Every session automatically
2. **Apply Intelligently** - AI determines relevance (recommended)
3. **Apply to Specific Files** - Pattern matching (*.test.ts)
4. **Apply Manually** - Activated with @-mention

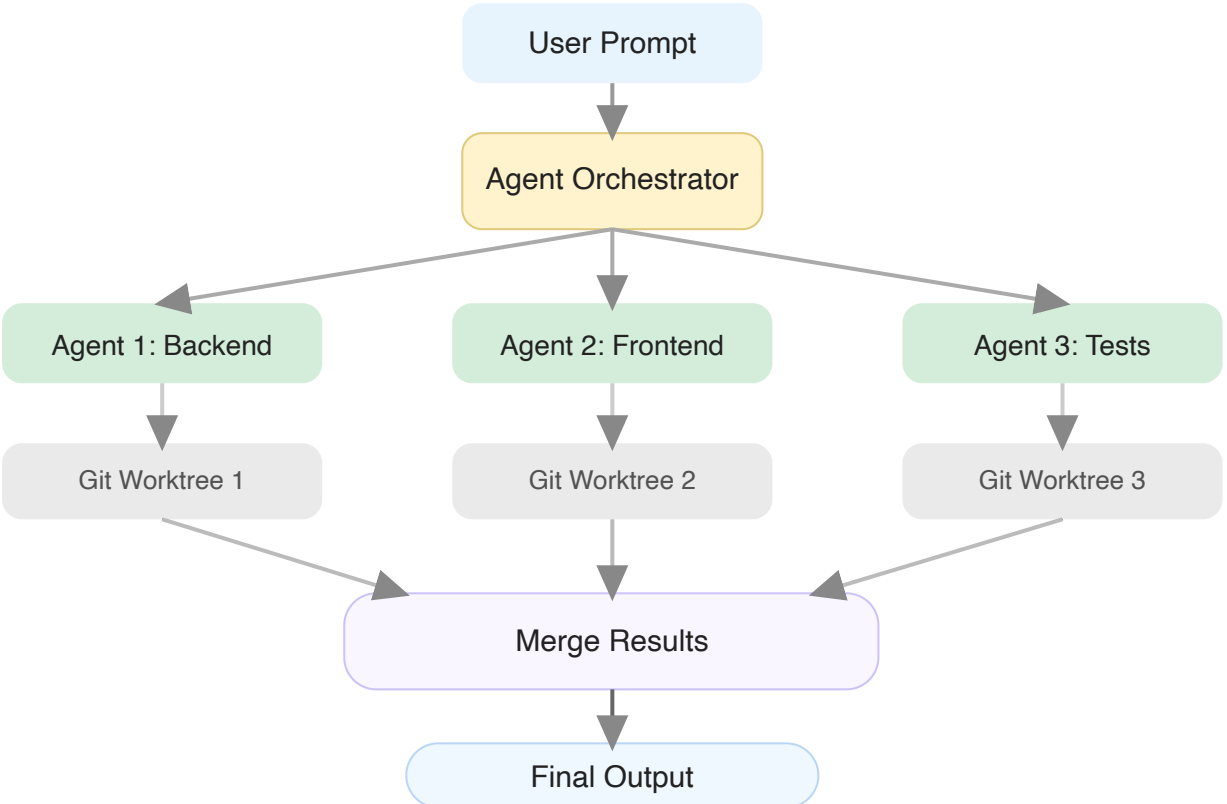
✅ When to Use

- Recurring mistakes
- Critical security patterns
- Team consistency

⚠️ Watch Out

- Can slow you down
- Keep under 500 lines
- Better context > more rules

Multi-Agent Architecture



Each agent operates in isolated git worktree → No conflicts

Model Context Protocol (MCP)

Open protocol connecting AI to tools & data

Like USB for AI—standardized interface

Model Context Protocol (MCP)

Open protocol connecting AI to tools & data

Like USB for AI—standardized interface

Tools

Functions AI executes

Prompts

Templated workflows

Resources

Structured data sources

GitHub MCP

- Create PRs
- Review code
- Manage issues

```
npx @modelcontextprotocol/server-github
```

Figma MCP

- Extract design tokens
- Generate components
- Sync UI updates

```
bunx cursor-talk-to-figma-mcp
```

GitMCP

- Turn any repo into MCP
- Replace github.com → gitmcp.io
- Instant context access

```
gitmcp.io/facebook/react
```

Database MCPs

- PostgreSQL
- SQLite
- Query and inspect

```
@modelcontextprotocol/server-postgres
```

[1]GitMCP | [2]Figma MCP

Demo: Cursor Configuration Setup

 See: `presentation/scripts.md`

Demo 1: Setting Up Cursor Configuration

Q&A - Section 1

Questions about Context Management?

💬 Chat features

📋 Project rules

🤖 Agent modes

🔗 MCP servers

Break

10 Minutes 

Building Your First Full App with Cursor

AI-Driven Project Initialization

1. Describe

Problem, users, features in plain English

→

2. Define

Tech stack, database, auth, deployment

→

3. Generate

Let Cursor scaffold everything

AI-Driven Project Initialization

1. Describe

Problem, users, features in plain English

→

2. Define

Tech stack, database, auth, deployment

→

3. Generate

Let Cursor scaffold everything

The Context File Approach

- Create `docs/context.md` with your vision
- Reference it inside prompts: `'Generate the project structure for @docs/context.md '`

[1] Building Apps with AI - Complete Workflow

Example: Context File

```
# DeepWork AI - Focus Productivity App

## Problem
Developers struggle with multitasking and distractions.
Need a tool to prioritize tasks and maintain deep work focus.

## Users
Software engineers, designers, product managers

## Core Features
1. Task management with priority levels
2. Focus timer for single-task concentration
3. AI chat to quickly add tasks via natural language

## Tech Stack
- Frontend: Next.js 14 (App Router), TypeScript, Tailwind
- Backend: Supabase (database + auth)
- AI: OpenAI API for chat
- Deployment: Vercel

## Database Schema
[Let Cursor generate based on features]
```

ChatGPT + Cursor Workflow

1️⃣ ChatGPT

Planning Phase:

- Brainstorm features
- Structure ideas
- Create context.md
- Plan architecture

→

2️⃣ Cursor

Implementation:

- Generate code
- Multi-file operations
- Debug errors
- Refactor

💡 **Use ChatGPT (or Claude/Gemini/etc.) for exploration, Cursor for implementation**

Demo: Building Your First Full App

Setting Up Supabase Backend

1. Create Project

- Visit supabase.com
- Create new project
- Copy API URL & anon key

2. Environment Variables

```
NEXT_PUBLIC_SUPABASE_URL=...
NEXT_PUBLIC_SUPABASE_ANON_KEY=...
```

3. Prompt Cursor

```
@docs/context.md

Install Supabase client and create
the database schema for tasks with:
title, description, priority,
deadline, completed status, user_id.
```

✅ **Cursor generates SQL migrations + RLS policies**

Creating UI Components

Prompt for Component Generation:

```
Create the following components following our project rules:

1. TaskCard - Display single task with priority badge
2. TaskList - Grid of TaskCards with filtering
3. AddTaskModal - Form to create new tasks
4. FocusTimer - Countdown timer for focused work

Use Tailwind, make them responsive, and include
TypeScript interfaces for all props.
```

Creating UI Components

✅ **Agent Mode Delivers:**

- TypeScript types
- Responsive design
- Accessibility
- Smooth animations

💡 **Pro Tip:**

Work file by file, review each component before moving to the next

Iterative Development Loop

1. Generate code with Agent or Composer

↓

2. Run the app - Check for errors: `npm run dev`

↓

3. Copy error messages - Paste into Cursor chat

↓

4. Let Cursor fix - Review the changes

↓

5. Commit - Save working state with git

[1] Complete App Building Workflow

Testing with AI Assistance

Generate Test Suites:

```
@src/components/TaskCard.tsx
  Create a complete test suite using Jest and React Testing Library.
  Include:
  - Rendering tests
  - User interaction tests
  - Edge cases
  - Accessibility tests
```

Testing with AI Assistance

Generate Test Suites:

```
@src/components/TaskCard.tsx
  Create a complete test suite using Jest and React Testing Library.
  Include:
  - Rendering tests
  - User interaction tests
  - Edge cases
  - Accessibility tests
```

Component Tests

Integration Tests

E2E Tests

Q&A

Questions about Building Your First App?

🏗️ Project initialization

🗄️ Data layer (Supabase)

⚛️ UI components

🧪 Testing strategies

Break

10 Minutes 

Advanced Architecture Patterns

🏗️ Atomic Design

- **Atoms** - Button, Input
- **Molecules** - SearchBar
- **Organisms** - Header
- **Templates** - Layouts
- **Pages** - Full views

🗃️ State Management

- **Local** - useState
- **Global Client** - Zustand
- **Server** - React Query
- **URL** - searchParams

🎣 Custom Hooks

- Single responsibility
- Named exports
- TypeScript interfaces
- Reusable logic

⚡ Performance

- React.memo()
- useMemo()
- useCallback()
- Code splitting

Demo: AI-Powered Quiz App

Quiz App: Tech Stack

AI-Generated Quiz Application

Frontend

Next.js 14
TypeScript
Tailwind CSS

AI

OpenAI API
Instructor
Structured Output

Backend

Supabase
Real-time
Row Level Security

Architecture Diagrams with Mermaid

Prompt Cursor to Generate Diagrams:

```
Create a Mermaid diagram showing the data flow in our quiz app:<br>
- User selects topic<br>
- API generates questions with OpenAI<br>
- Questions displayed one at a time<br>
- Answers stored in Supabase<br>
- Results calculated and shown
```

Architecture Diagrams with Mermaid

Prompt Cursor to Generate Diagrams:

```
Create a Mermaid diagram showing the data flow in our quiz app:<br>
- User selects topic<br>
- API generates questions with OpenAI<br>
- Questions displayed one at a time<br>
- Answers stored in Supabase<br>
- Results calculated and shown
```

✅ Diagram Types

- flowchart - Logic flow
- sequenceDiagram - Interactions
- classDiagram - Data structures

💡 Best Practice

Start small, layer upward, multiple perspectives

[1] Cursor Cookbook - Mermaid Diagrams

Figma MCP Integration

Figma Design

- Design tokens
- Component specs
- Typography

→

React Code

- Pixel-perfect
- Tailwind classes
- Responsive

Prompt Example:

```
@figma file:quiz-app-designs

Generate the QuestionCard component matching the Figma design.
Extract all design tokens (colors, spacing, shadows) and
create a matching React component with Tailwind.
```

[1] Cursor Talk to Figma MCP

Q&A

Questions about Development Patterns?

🏗️ Architecture patterns

🎨 Figma integration

📊 Mermaid diagrams

⚡ Performance tips

Full-Stack Application

Backend API Development

Prompt for Complete REST API:

```
Create a complete REST API for our quiz app:

Endpoints:
- POST /api/quiz/generate - Create new quiz
- GET /api/quiz/[id] - Get quiz by ID
- POST /api/quiz/[id]/answer - Submit answer
- GET /api/quiz/[id]/results - Get final results
- GET /api/quiz/history - User's past quizzes

Include: Authentication, validation (Zod), error handling,
rate limiting, API documentation comments
```

Backend API Development

Auth

Validation

Rate Limiting

Database Optimization

Optimization Strategies

- **Indexes** - Speed up queries
- **JOINs** - Reduce round trips
- **Pagination** - Limit results
- **Caching** - Redis layer
- **Connection Pooling** - Supavisor

Prompt Cursor:

```
@src/lib/db/queries.ts

Optimize this query for performance:
- Add proper indexes
- Use JOIN instead of multiple queries
- Implement pagination
- Add caching strategy
```

State Management & Real-time

Zustand Store

Quiz state, answers, timer

Optimistic Updates

Instant UI feedback

Real-time Sync

Supabase subscriptions

Optimistic Update Pattern

1. Update UI immediately (optimistic)
2. Send request to server
3. If error: rollback UI changes
4. If success: keep UI as-is

Result: App feels instant ⚡

Demo: Test, Iterate, Deploy

Q&A

Questions about Full-Stack Development?

🏗️ System architecture

🗄️ Database optimization

⚡ State management

🔄 Real-time features

Break

10 Minutes 

Deployment & Production

Deployment with Vercel

From Code to Production in 10 Minutes

Zero Config

Auto optimizations

Edge Network

Global CDN

Instant Rollback

One-click revert

CI/CD Pipeline

Generate GitHub Actions Workflow:

Create a GitHub Actions workflow for our Next.js app:

On every PR:
1. TypeScript type checking
2. ESLint
3. Tests (unit + integration)
4. Build
5. Deploy preview to Vercel

On merge to main:
All above + E2E tests + Production deploy + Slack notification

CI/CD Pipeline

✅ **Type Check**

🧪 **Test**

🚀 **Deploy**

Quality Gates

1. Type Safety

- strict: true
- noImplicitAny
- noUncheckedIndexed

2. Linting

- ESLint rules
- Accessibility checks
- Import ordering

3. Test Coverage

- 80% threshold
- Branch coverage
- Function coverage

4. Security

- npm audit
- Snyk scanning
- Secret detection

5. Performance

- Bundle size limits
- Lighthouse CI
- Web Vitals

6. Monitoring

- Sentry errors
- Vercel Analytics
- Database metrics

Production Readiness Checklist

🔐 Security

- ✅ Environment variables secured
- ✅ SSL certificates valid
- ✅ CORS policies set
- ✅ Rate limiting enabled
- ✅ SQL injection prevention

⚡ Performance

- ✅ CDN configured
- ✅ Image optimization
- ✅ Code splitting
- ✅ Caching strategy
- ✅ Database indexes

📊 Monitoring

- ✅ Error tracking (Sentry)
- ✅ Analytics setup
- ✅ Uptime monitoring
- ✅ Database metrics
- ✅ Alert system

💾 Backup & Recovery

- ✅ Automated backups
- ✅ Backup verification
- ✅ Recovery runbook
- ✅ Rollback strategy
- ✅ Feature flags

Automated PR Workflow

Staff Engineer Tip: Handle PR Feedback Fast

1. Screenshot

Capture PR comments

→

2. Cursor Plan

Drag into Cursor, generate plan

→

3. Execute

Fix items one by one

[1] Automating PR Feedback Workflow

Demo: Vercel & Github Actions

Cost Optimization Strategies

💸 Current Costs

- Vercel Pro: $20/month
- Supabase Pro: $25/month
- OpenAI API: ~$50/month
- Upstash Redis: $10/month

Total: ~$105/month

💰 Optimizations

- ✅ Switch to DeepSeek (10x cheaper)
- ✅ Aggressive caching
- ✅ Free tier for previews
- ✅ Optimize DB queries
- ✅ Rate limiting

New Total: ~$40/month

💡 Prompt Cursor for Cost Analysis

Analyze our app's costs and suggest optimizations to reduce spending while maintaining performance.

Key Takeaways

✅ Context is King

Strategic @-mentions, clean project structure

✅ Three-Step Framework

Explore → Plan → Build

✅ Production-Ready

Testing, CI/CD, monitoring, security

✅ MCP Integration

GitHub, Figma, databases

🎯 Remember

Cursor augments your workflow—it doesn't replace engineering judgment. Use it to eliminate tedious work and focus on what matters.

Additional Resources

📚 Documentation

- Cursor Docs
- Changelog
- 2.0 Announcement

🔗 MCP Servers

- Official Servers
- GitMCP
- Figma MCP

📹 Videos

- Staff Engineers
- Building Apps

Connect With Me

📚 Course materials

🔗 LinkedIn

🐦 Twitter/X - @LucasEnkrateia

📺 YouTube - @automatalearninglab

📖 Blog

📧 Email: lucasenkrateia@gmail.com