Chapter 4: Your First AI-Built App in One Hour

What you're about to do: Describe what you want in one comprehensive prompt. Watch AI build it. Deploy it live. That's it.

Time: 60 minutes total

- Setup: 10 minutes
- Understanding the prompt: 5 minutes
- Building: 30-40 minutes
- Deploy: 5 minutes

What you'll build: A personal bookmark manager that saves links, adds descriptions, and organizes them with tags.

The shift: You're not following 28 steps. You're describing what you want once, then letting AI figure out the how.

Part 1: Setup (10 Minutes)

You need three accounts and one download. That's it.

Create These Accounts:

1. **Supabase** (supabase.com) - Your database
2. **Vercel** (vercel.com) - Your hosting
3. **Cursor** (cursor.sh) - Your AI coding tool

All free tiers. Sign up with GitHub for fastest setup.

Install Cursor:

Download from cursor.sh and install. It looks like VS Code because it's built on top of it.

That's your setup. No environment variables yet. No configuration files. We'll handle those when AI asks for them.

Part 2: The Master Prompt (5 Minutes)

Here's what most people don't understand: **One comprehensive prompt can replace 28 step-by-step instructions.**

The difference is specificity. Not "build a bookmark app." Not even "build a bookmark app with tags and search."

You describe:

- What it does
- What the user sees
- How data flows
- What tech to use
- What constraints to follow

The Bookmark Manager Master Prompt

Copy this entire prompt. This is what you'll paste into Cursor:

```
GOAL: Build a personal bookmark manager web app

WHAT IT DOES:
- Save website URLs with descriptions
- Add multiple tags to each bookmark
- View all bookmarks in a clean list
- Search bookmarks by title, description, or tags
- Click a bookmark to open in new tab
- Delete bookmarks I no longer need

USER EXPERIENCE:
- Landing page with input form at top (URL field, description field, tag input)
- Below form: list of all saved bookmarks
- Each bookmark shows: title (from URL), description, tags as pills, timestamp
- Click bookmark title to visit site
- Each bookmark has a delete button
- Search bar filters bookmarks in real-time
- Clean, minimal design (like Notion or Linear)

TECHNICAL REQUIREMENTS:
Stack:
- Next.js 14 with App Router
- TypeScript
- Tailwind CSS for styling
- Supabase for database

Database Schema:
- Bookmarks table: id, url, title, description, tags (array), created_at
- Use Supabase's built-in auth (optional for v1, but set up the structure)

Features:
- Auto-fetch page title from URL when saving
- Tag input with autocomplete from existing tags
- Responsive design (works on mobile)
- Loading states for all actions
- Error handling with user-friendly messages

CODING CONSTRAINTS - FOLLOW THESE RULES:

BEFORE YOU CODE:
1. Summarize what you understand about the requirement
2. Ask specific questions about any uncertainties
3. List files you plan to create and why
4. Get confirmation before proceeding

WHILE CODING:
1. Make ONLY the changes needed for the specific requirement
2. Do NOT add features, enhancements, or "improvements" not requested
3. Do NOT refactor or reorganize code while building
4. If you modify the same file 3+ times, STOP and explain what you're trying to do
```

```
WHEN COMPLETE:
1. Verify the specific requirement works
2. Test all user actions (add bookmark, search, delete)
3. Provide clear deployment instructions

Start by confirming you understand this project, then ask any clarifying questions before
generating code.
```

What Makes This Prompt Work

Goal clarity: First line states exactly what we're building

User perspective: Describes the experience, not just features

Technical specificity: Exact stack, schema, and requirements

Constraints included: The coding constraints prevent AI from over-building

Control maintained: AI must confirm understanding before coding

This is the formula. Goal → Experience → Technical → Constraints → Confirmation.

Part 3: Building (30-40 Minutes)

Step 1: Start the Conversation

1. Open Cursor
2. Create a new folder: `bookmark-manager`
3. Open that folder in Cursor
4. Open the chat panel (Cmd+L or Ctrl+L)
5. Paste the entire master prompt

What happens next: AI will read everything, then respond with:

- A summary of what it understood
- Clarifying questions (maybe 2-5)
- A list of files it plans to create

Step 2: Answer Questions

AI might ask:

- "Should the tag autocomplete be case-sensitive?"
- "Do you want tags to be created on-the-fly or pre-defined?"
- "Should there be a limit on number of tags per bookmark?"

Your job: Answer honestly. Don't overthink it. "Case-insensitive is fine," "On-the-fly," "No limit needed."

If you don't know: "Your call, make it user-friendly."

Step 3: Give Permission to Build

Once questions are answered, AI will show you a file plan:

```
I'll create:
- app/page.tsx (main UI)
- app/api/bookmarks/route.ts (API endpoints)
- lib/supabase.ts (database connection)
- components/BookmarkForm.tsx
- components/BookmarkList.tsx
- components/SearchBar.tsx
```

You type: **"Looks good, proceed."**

Step 4: Watch It Build

AI will generate all files. You'll see them appear in your file tree.

Don't panic. You don't need to read every line right now. The point is to see it work first.

Step 5: Set Up Supabase

Cursor will hit a pause point: "I need your Supabase credentials."

1. Go to your Supabase project
2. Settings → API → Copy these two values:
   - `Project URL`
   - `anon/public key`
3. Create `.env.local` in your project root
4. Paste this template with your actual values:

```
# Supabase Configuration
NEXT_PUBLIC_SUPABASE_URL=https://your-project.supabase.co
NEXT_PUBLIC_SUPABASE_ANON_KEY=your-anon-key-here

# IMPORTANT:
# - Never commit .env.local to git (it's in .gitignore by default)
# - Use different values for production (add in Vercel dashboard)
# - Restart dev server after adding/changing variables: Ctrl+C, then npm run dev
```

Tell Cursor: "Added the environment variables."

Step 6: Create the Database Table

AI will give you SQL to run. Copy it.

1. Go to Supabase → SQL Editor
2. Paste the SQL
3. Run it
4. Tell Cursor: "Table created."

Step 7: Run It Locally

Cursor will tell you to run:

```
npm install
npm run dev
```

Do it. Open `localhost:3000` in your browser.

Step 8: Test Everything

Try these actions:

- Add a bookmark with a URL
- Add some tags
- Add another bookmark
- Use the search
- Delete a bookmark

If something breaks: Copy the error, paste it in Cursor chat, say "This error appeared, fix it."

This is the magic moment. You described what you wanted 30 minutes ago. Now you're using it.

Part 4: Deploy It Live (5 Minutes)

Make It Public

1. Push to GitHub:

```
git init
git add .
git commit -m "Built with AI"
git remote add origin YOUR_GITHUB_REPO_URL
git push -u origin main
```

2. Go to Vercel → New Project → Import your GitHub repo

3. Vercel will detect Next.js automatically

4. Add your environment variables in Vercel:

   - `NEXT_PUBLIC_SUPABASE_URL`
   - `NEXT_PUBLIC_SUPABASE_ANON_KEY`

5. Click Deploy

In 2-3 minutes: You have a live URL. Send it to friends. Use it on your phone. It's real.

Part 5: What You Just Did

Let's reflect on what actually happened here.

The Old Way vs The New Way

Old way (traditional coding):

1. Learn React
2. Learn Next.js
3. Learn Tailwind
4. Learn Supabase
5. Then build

New way (what you just did):

1. Describe what you want
2. AI builds it
3. You learn by using it

You Didn't Write Code, But You Built Software

What you actually did:

- Defined requirements clearly
- Made architecture decisions (Next.js, Supabase, Tailwind)
- Answered design questions (how tags work, what UI looks like)
- Tested user experience
- Debugged errors
- Deployed to production

That's product building. The code was the implementation detail.

Understanding What AI Built

You don't need to read every line, but you should understand the structure:

app/page.tsx: Your main UI - the form and bookmark list **app/api/bookmarks/route.ts:** Handles save/delete actions **lib/supabase.ts:** Connects to your database **components/:** Reusable UI pieces

Open `app/page.tsx`. You'll recognize things: the form fields, the button, the list. It's React, but you can read it.

The confidence shift: You went from "I don't know React" to "I can see how this React app works."

How to Modify It

Want to add a feature? Use the same approach:

Prompt:

```
Add a feature: When I save a bookmark, automatically fetch and save
the page's favicon (small icon). Display it next to each bookmark
in the list.

Follow the same coding constraints as before. Ask questions if unclear.
```

AI will ask questions, explain what files to modify, make changes.

You're not coding. You're directing.

What This Chapter Actually Taught You

The One Prompt Method

Not: "Build this, then do this, then do that, then..." **But:** "Here's everything I want. Ask questions. Build it."

This is the shift. Comprehensive prompts replace step-by-step tutorials.

The Builder Mindset

You just:

- Defined a product
- Made tech choices
- Managed AI as a developer
- Tested user experience
- Shipped to production

That's the full cycle. You're not "learning to code." You're building products with AI as your implementation partner.

The Confidence You Gained

Before this chapter: "I need to learn Next.js and React and Supabase and..."

After this chapter: "I can describe what I want and get a working app."

That's not arrogance. That's reality. You just proved it to yourself.

Next Steps

Modify Your Bookmark App

Try these changes (each is one prompt):

1. **Add folders:** Organize bookmarks into folders/categories
2. **Add export:** Button to export all bookmarks as CSV
3. **Add sharing:** Generate a public link to share specific bookmarks
4. **Add dark mode:** Toggle between light and dark themes

Each one is practice. Each one builds confidence.

Try a Different App

Use the same master prompt structure for:

- **Expense tracker:** Log expenses, categorize, view totals
- **Habit tracker:** Check off daily habits, see streak counts
- **Recipe saver:** Save recipes with ingredients and instructions

Same pattern:

- Goal
- User experience
- Technical requirements
- Coding constraints
- Build it

Understand the Tools Deeper

Now that you've used the stack:

- Read Next.js docs (you'll understand them now)
- Explore Supabase features (realtime, auth, storage)
- Learn Tailwind patterns (you've seen them in your code)

The difference: You're learning by extending something real, not building todo apps from tutorials.

The Unlock

This chapter was different from traditional coding tutorials on purpose.

Traditional tutorial: "Here's 28 steps. Follow them exactly. Type this. Type that. Now you have an app."

What we just did: "Here's a comprehensive prompt. Describe what you want. Let AI build it. Now you have an app."

The result is the same: Working software in production.

The feeling is different: You feel capable, not hand-held.

That's the confidence shift. That's vibe coding.

You didn't learn syntax. You learned to build.

Connect & Share

💌 **Newsletter**: Build to Launch - Weekly AI building tips, templates, and real builder stories

✍️ **Medium**: AI Builders - Read more articles and guides

💬 **Reddit**: r/VibeCodingBuilders - Join the community

🦋 **Bluesky**: @jenny-ouyang - Connect

💼 **LinkedIn**: Jenny Ouyang - Connect