Vibe Coding (Part 1)

Project at a Glance:

- **The Goal:** Build a fully interactive, clickable web app to turn your child's idea into a working tool.
-
- **The Tool:** Anthropic's Claude (free tier available, no coding experience required).
-
- **Who Drives?:** Perfect for a wide age range (7–18). Younger children provide the idea; parents execute. Older children can take the keyboard.

Why I Built My Own App - and You Can Too

A while back, my nutritionist suggested that I should consider trying an anti inflammatory diet. New food restrictions, new recipes, and suddenly a constant nagging question every time I stepped into the kitchen: "Is this ingredient actually okay to eat?" The same searches kept happening over and over.

I found myself searching the same question over and over. Is potato anti-inflammatory? What about quinoa flour? Can I swap this ingredient for something better? It was inefficient. So I decided to just build a tool that would answer all of it in one place: an app where I could check any ingredient, input what I had in the fridge to get recipe ideas, analyze an existing recipe to get smart substitution suggestions on the spot.

If I had built this from scratch in code, it would have taken weeks. With AI as my co-builder, I had a working first version within an hour.

Children have similar repeating questions and mini problems in their own worlds: logging reading time, organizing a community garden, or running a tiny fundraiser. The same approach that turned my kitchen frustration into an app can turn their ideas into working tools, and that is the focus of this note.

HEALTHY DIGITAL CHILDHOOD ALLIANCE

Introduction: The “Syntax” Barrier Is Gone (Vibe Coding)

Learning to build an app or a video game used to mean a steep learning curve getting fluent in complex computer languages like Python or JavaScript. The barrier is now significantly reduced.

Over the last year, a major shift has occurred in the tech industry. It is called “Vibe Coding.” You no longer need to speak the computer’s language. You simply describe what you want the app to do in plain English, the “vibe,” and the AI writes the code and builds the app right in front of you.

But here is the critical distinction: AI is a fast “try it and see” tool. It helps you turn an idea into something you can actually touch and test in minutes instead of months. The vision, what problem to solve, how success is measured, what the app should do, and the experience users have, comes entirely from the human. The quality of what gets built is only as good as the clarity of the idea behind it.

This is why, before we start vibe coding, we will walk through a three-phase process used by real product designers and startup builders to turn a vague idea into a precise brief. The human leads. The AI follows.

HEALTHY DIGITAL CHILDHOOD ALLIANCE

The Landscape: Meet Claude

Many families have already encountered general purpose AI tools such as ChatGPT, Microsoft Copilot, or Google Gemini. For this project, the focus is on Claude, an AI system built by Anthropic.

Claude is particularly well suited to building small web apps with children because of a feature called Artifacts. When a user asks Claude to build something such as a web page, a game, or a small tool, it not only writes the code but also renders a live preview in a panel next to the chat. Buttons can be clicked, forms filled out, and behavior tested instantly, all inside the browser, without installing any extra software.

Artifacts were first introduced with Claude 3.5 Sonnet in mid 2024 and have become a core part of Anthropic's product. As of early 2026, Claude Sonnet 4.6 is the default model on claude.ai, with faster performance and stronger coding capabilities while keeping the same Artifacts experience.

There are also dedicated vibe coding platforms such as Lovable, V0, and Base44, which take users from idea to published app in a single visual environment. These tools are powerful, but they tend to hide the code and focus on fast launch, making debugging challenging. Claude offers a balance: quick interactive previews plus transparent access to the underlying code.

A key detail for older children is the Preview / Code toggle in the Artifacts panel. At any time, they can switch between the live app and the full HTML, CSS, and JavaScript that power it. This makes the tool not just a black box that "spits out apps" but also a window into how professional grade code is structured.

Getting Started: Families can create a free account at claude.ai with email verification and use Artifacts directly in the browser.

Age Guidelines: Who Drives?

The core learning goal is the same for everyone: turning an idea into clear instructions, testing the result, and improving it. But who drives depends on your child's age and experience with AI tools.

A note on age guidelines: in the Capabilities Phase of this series, I assume your child has had some AI literacy foundation, whether through this series, school, or dinner table conversations, and understands the basics of how AI works and how to stay safe online. If that foundation is not yet in place, start one level up in supervision until they are comfortable with the tool and your family's boundaries around it.

Under 13 Co-Use (Parent as Operator):

Your child is the Product Manager, deciding what the app does, what buttons say, and what to track. The parent controls the screen, types the prompts, and handles the account. The child learns the full process of going from idea to execution without the frustration of wrestling with interfaces.

Ages 13–15 Active Mediation:

Your child can drive the tool, but a parent stays close, co-piloting sessions, reviewing prompts regularly, and keeping the conversation open. Make sure the core thinking happens together before your child touches the keyboard. Younger teens are still building judgment and more prone to skipping the thinking and jumping straight to building.

Ages 16–18 Guided Independence:

Your teen drives with parents periodic check-ins. Agree on clear boundaries around the project scope and review what they have built together at the end of each session. Supervision shifts from constant to periodic, not to zero.

- **The Project: Turn Your Idea Into an App**
-
- The project is straightforward: help a child turn a real idea, something they care about, into a working app. It might be a hobby tracker, a mini business tool, a community project, or a way to understand something in their own life.
-
- The process mirrors how real product teams work. The human defines the vision; the AI helps execute it. To keep the thinking front and center, this note uses three structured phases, each tied to a framework used in professional settings.
-
- **Phase 1: The Job: What Is Actually Frustrating? (Jobs to Be Done)**
-
- Product teams often talk about "Jobs to Be Done": the idea that people do not buy or use tools for their own sake but to get a job done more easily, quickly, or reliably. The starting point for any app is understanding what job the user is already trying to do and what feels frustrating about it today.
-

Sit down with your child and answer two questions on paper:

-

12. **What job are you trying to get done?**
    - For example: "Every time we go to the beach, we pick up trash. We want to track the impact we are making and, over time, see that it actually adds up."
2. **What is frustrating about how you do it today?**
    - For example: "We do the cleanups but do not track anything right now. It would be nice to have a way to record what we collect, watch our impact grow over time, and maybe even encourage others to do the same, or at least not litter."
    -
    - Frustration does not have to be about something broken; it can be about something invisible or unmeasured. In this case, the job is emotional and social: feeling that your effort matters and being able to show evidence of that effort to others.
    -
    - If no existing tool helps a family record and visualize casual beach cleanups in a simple, shared way, that gap becomes the seed of the app.

Phase 2: Define and Ideate: What Could Solve It? (Design Thinking)

Design Thinking is a structured way of moving from observation to clear problem statements to possible solutions. It is a problem-solving method taught at Stanford and used in schools and companies worldwide. Jobs to Be Done helped uncover the real job and frustration; the Define step in Design Thinking sharpens that insight into a single, clear problem statement that can guide design work.

Define the problem in one sentence.
Taking the beach cleanup example:

Currently, kids are not measuring the impact of their beach cleanups, so it is hard to see that impact grow over time or share it with others in a motivating way.

Ideate: explore at least three possible solutions before picking one.

Consider options together:

- Could a shared notes app solve this? It is easy to set up, but hard to scale beyond a small group, and it does not calculate totals or trends the way a spreadsheet or custom app can.
-
- Is there an existing app that already does most of this? A quick search for “litter tracking” or “beach cleanup” apps can show what exists and where it falls short.
-
- What would a custom app do differently? For example, make it effortless for any participant to log a cleanup session, attach a photo, and see both personal and community totals.

The custom app might allow users to enter the beach name, date, a photo, total weight collected, and a short note about what types of trash is picked up. Over time, it would show totals per beach and across all beaches, turning scattered efforts into a visible story. If no existing tool does exactly this in a child friendly, free way, that is the gap the app will fill.

Only after this conversation should the decision be made to build. By this point, the child has a real job, a clearly defined problem, and a distinct idea. That is the same kind of brief product designers and founders work from.

Phase 3: The Lean Loop: Build, Measure, Learn (Lean Startup)

In practice, most successful software does not emerge from a perfect plan. It evolves through a loop known as Build - Measure - Learn: shipping a minimum version, testing it, and using the results to decide what to change next. For children, this loop turns trial and error into a structured habit.

HEALTHY DIGITAL CHILDHOOD ALLIANCE

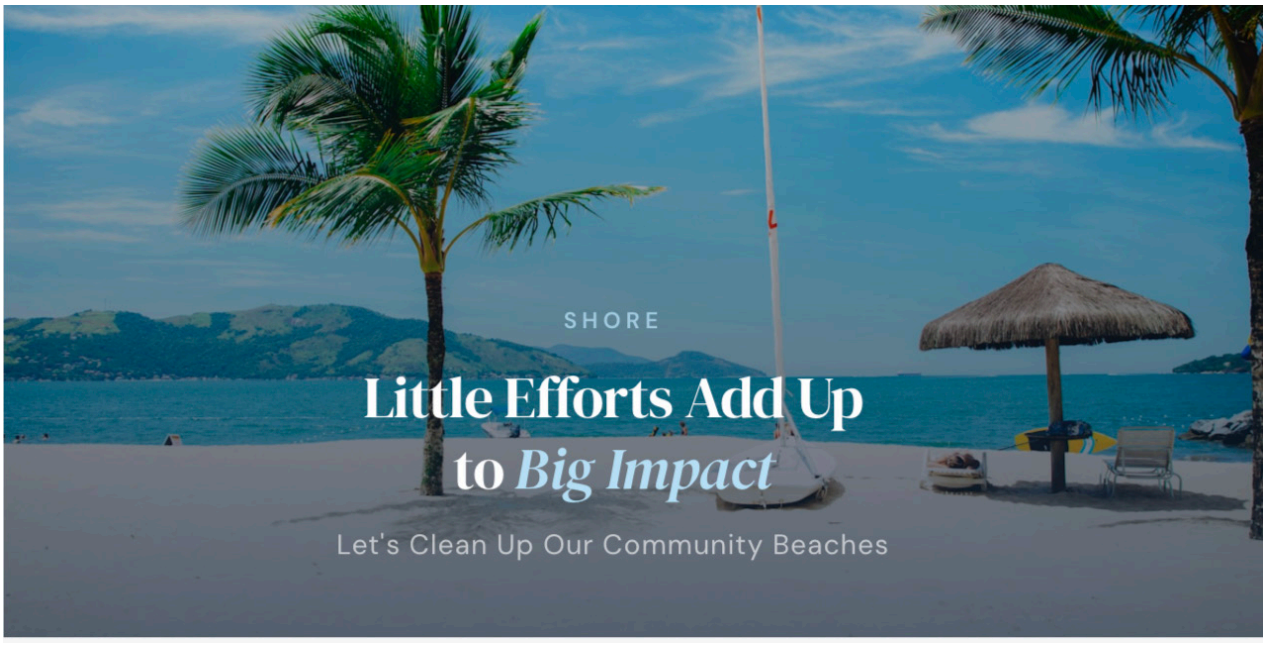
Build: Write the Prompt and Launch

Open Claude at claude.ai and use a simple template to describe the app in precise terms:

“Please build a colorful, interactive web app to [describe the job from Phase 1]. Here are the inputs: [Input 1], [Input 2], …. Here are the rules: Rule 1: [describe rule]. Rule 2: [describe rule]. Please make it visually appealing with bright colors and large, clickable buttons.”

For the beach cleanup tracker, the prompt could be:

“Please build a fun, interactive web app to track our beach cleanup impact. Here are the inputs: beach name, date of cleanup, photo upload of items collected, total weight in kilograms, and a free text field for notes. Here are the rules: show a log of all cleanup sessions with the photo, date, beach name, and weight for each entry; calculate the total weight collected across all sessions; show a breakdown by beach so we can see how much has been collected from each one over time; and when the total weight across all sessions hits 10 kg, show a celebration animation. Please use a bright blue and sandy color theme with big, friendly buttons.”



HEALTHY DIGITAL CHILDHOOD ALLIANCE

Talk Back to Claude and Improve

Every issue found in testing becomes a new, more precise instruction. Instead of saying, “It does not work”, the child can say:

“When I add three sessions, 2 kg, 3 kg, and 5 kg, the total only shows 5 kg instead of 10 kg. Please fix the calculation so the total equals the sum of all entries.”

Claude then revises the code and updates the Artifact preview. Children repeat the loop: build, test, observe, and refine. Over time, they practice debugging not just by trial and error but through increasingly specific language.

The Three Frameworks Behind the Project

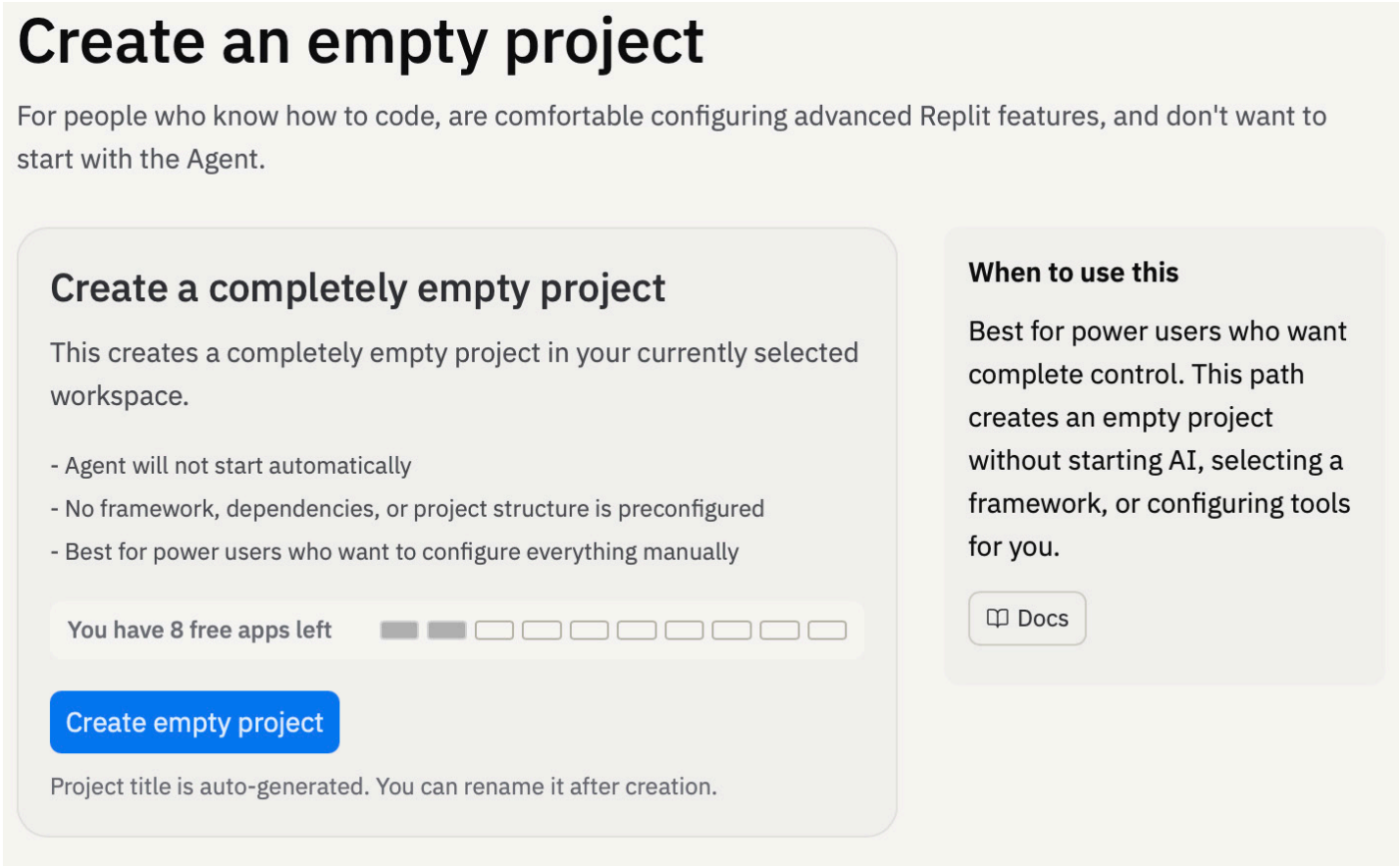
The phases above are not arbitrary; they mirror methods used in professional product development.

Phase	Framework	Core Question
Phase 1: The Job	Jobs to Be Done	What job is the app being "hired" to do, and what is frustrating today.
Phase 2: Define and Ideate	Design Thinking	What is the precise problem, and what are the possible solutions.
Phase 3: Build - Measure - Learn	Lean Startup	What is the smallest version we can test now, and what do we change based on what we learn.

These are not abstract classroom concepts. They are used every day by designers, entrepreneurs, and engineers to ship real products. Introducing them early helps children see app building as a thinking and problem solving process they can use in many areas of life, not just in technology.

- **From Prototype to Real Life: Sharing the App**
-
- After a few Build - Measure - Learn cycles, there will be a prototype that behaves consistently and feels right for the family. The next question is how to move from a private preview inside Claude to something that can be shared more widely.
-
- **Step 1: Download and Test with Family and Friends**
-
- When you are happy with how the app looks and behaves in Claude, click the Download button to save the HTML file to your device. You can open this file in any modern web browser directly. Share this file directly with family and friends for initial testing. The HTML file is safe to share this way: it runs entirely in the browser and stores no personal data. Use this round of testing to answer: Does the layout feel right on a phone? Are the buttons easy to tap? Can someone navigate it without explanation? Go back to Claude, describe what needs to change, and download the updated version. Repeat until it feels right.
-
- One important limitation: this version does not save data. If you close the browser, your logged cleanups are gone. Treat it purely as a way to perfect the design before going live.

- **Step 2: Add Data Persistence and Publish as a Live Site with Replit**
-
- Before moving the app out of Claude, it is worth adding a way for the app to remember past entries (data persistence) so they do not disappear when the page is refreshed or the browser is closed. Modern browsers can store small amounts of information directly on the device, so the app can load your previous data the next time it opens. To do this, input a prompt like the following in Claude:
-
- *“Please update this app so that it saves all entries in the browser using localStorage and reloads them whenever the page opens.”*
-
- Now the app is ready to move to a hosting environment. Replit is a browser-based platform for turning a finished app into a live website. Since you have already built and tested the app in Claude, the best path is to create an empty project and paste in Claude’s final code.



- **A typical flow looks like this:**
-

3. In Claude, ask it to add data persistence as described above, and verify that entries remain after a page refresh.
4. Then ask “Please give me the complete final code for this app, including the persistence changes.”
5. Copy the code from the Artifacts panel.
6. Go to replit.com, sign in or create a free Replit account, and choose Import code or design and then Create a completely empty project.
7. Add the Claude code into the files Replit creates for you.
8. Use Replit’s Publish button to host the app and obtain a public URL that you can share.

-
- This approach preserves the look and logic that were refined inside Claude, while adding a stable, shareable link other people can visit from their own devices.
-
- **Step 3 (Optional): Add a Custom Domain**
-
- Replit assigns each project a generated URL, you can’t select what it is. For families who want an easy to remember address such as beachcleanup.co can register a custom domain name with providers like GoDaddy or Netlify’s domain service for a recurring fee. Once a domain is purchased, it can be connected to the hosted app so that visitors see the custom name in their browser. For most family projects, this step is optional and can wait until the app gains traction.
-
- Lovable, V0, Base44, and similar vibe coding platforms offer alternative pathways where building and publishing happen in one integrated environment. Those are useful to know about, especially for older teens exploring different tools, but for families who have already invested time iterating in Claude, Replit provides a natural continuation of that work.

- **What Is Your Child Actually Learning?**
-
- A common parental concern is that if the AI is writing the code, children might not be learning anything substantial. In practice, short projects like this exercise the skill of computational thinking rather than memorization of syntax.
-
- Computational thinking involves breaking a big, messy idea into smaller steps, identifying patterns, designing rules, and specifying clear instructions that a computer or AI can follow. When a child skips Phase 1 or Phase 2 and jumps straight to “Build me an app”, the result is often an interface that looks plausible but does not solve the original problem; they soon discover that their thinking was incomplete.
-
- The mental workout has not disappeared; it has shifted. Instead of learning where semicolons go, children learn to:
-

9. Clarify goals and metrics for success.
10. Translate real world processes into discrete inputs and rules.
11. Inspect outputs critically and communicate precise corrections.

-
- This matches how many education researchers and professional organizations now describe the role of computational thinking in the age of AI: less about turning every student into a full time programmer and more about equipping them to collaborate effectively with powerful automated systems.
-
- Earlier in this note, AI was described as a fast "try it and see" tool. That idea ties directly into computational thinking: AI makes it cheap to test whether a set of rules and inputs really captures the intended behavior. Children design the logic, AI runs the experiment, and the child decides what to adjust.

- **Going Further (Optional)**
-
- Some children will finish one app and immediately want to build something more complex. When that happens, upgrading within the Claude ecosystem can unlock more ambitious projects.
-
- **Claude Pro**
- Claude Pro is Anthropic's paid subscription for heavier everyday use. It costs about 20 USD per month and raises the usage and context limits compared with the free tier, so Claude can remember longer conversations and larger projects without hitting a wall. It is a good next step once your child is regularly working on multi-screen apps or teaming up with classmates and you find the free plan's limits getting in the way.
-
- **Claude Code**
- Claude Code is Anthropic's advanced coding environment for Pro and Max users. Instead of working on a single file at a time, it can read an entire project, make coordinated changes across many files, run tests, and fix problems from a single natural-language request. This is aimed at older teens or parents who are comfortable looking at real code and want to build larger, more "real world" software projects, not just single-page apps.
-
- **Claude API tokens**
- For families or advanced teens who want their apps to talk to other services (for example, Google Sheets or a school database) or to run on their own servers, Anthropic offers a developer API. Instead of a flat subscription, you pay only for what you use, based on the amount of text the model processes, measured in "tokens." This pay-as-you-go model makes sense when Claude is one component inside a larger custom system, rather than the main place where your child chats and builds.
-
- In all cases, the recommendation is to exhaust what is possible on the free claude.ai tier first. Many rich learning experiences, especially for children in the 7 to 14 range, fit comfortably within those limits. Consider Pro only when your child consistently runs into usage limits, and treat Claude Code and the API as tools for older teens who are clearly pulling toward deeper technical work.