Email Tone Checker - C4 Documentation

Student: Surabhi Lone(surabhi.lone@gmail.com)
Position: Student Assistant (m/f/d): GenAI-Incubator
Date: August 21, 2025

---

1. **Level 1 : System Context Diagram**
   What This Shows : This is the top view of my system. I wanted to understand who would use my Email Tone Checker and what external services it would need to work with.

   How I Thought About It : The core idea is simple: someone writes an email that might not sound quite right, my system helps them improve it using AI, and they get back suggestions they can actually use. The only external service I need is Google's AI to do the smart analysis part.

   The Key Relationships : When I mapped this out, I realized there are really just three main actors:
   1. The person writing emails (my users)
   2. My web application (the system I'm building)
   3. Google's AI service (the smart part I can't build myself)

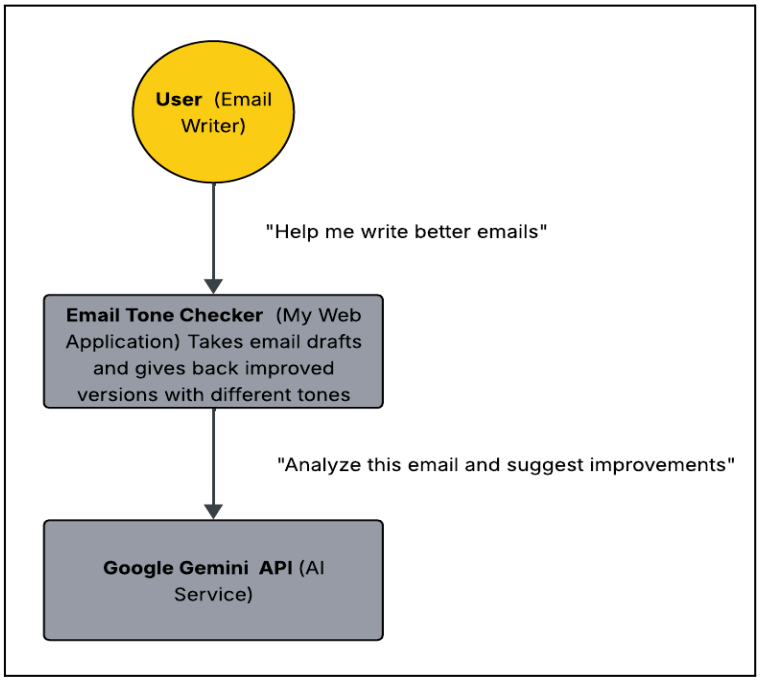

Fig 1. Context Diagram

2. **Level 2 : Container Diagram**

What I Decided to Build : When I started thinking about implementation, I realized I needed three main pieces working together. I chose technologies based on what would be most reliable and easiest to maintain.

My Architecture Decisions : I made several key decisions here:

1. Frontend in the Browser: I wanted something that works for everyone, so I stuck with standard HTML, CSS, and JavaScript. No fancy frameworks that might break or require users to download anything.
2. Serverless Backend: Instead of managing my own server, I used Netlify Functions. This means I don't have to worry about server maintenance, and it automatically scales if lots of people use it.
3. External AI Service: Rather than trying to build my own AI (which would take years), I use Google's Gemini API. It's powerful, reliable, and much smarter than anything I could create.

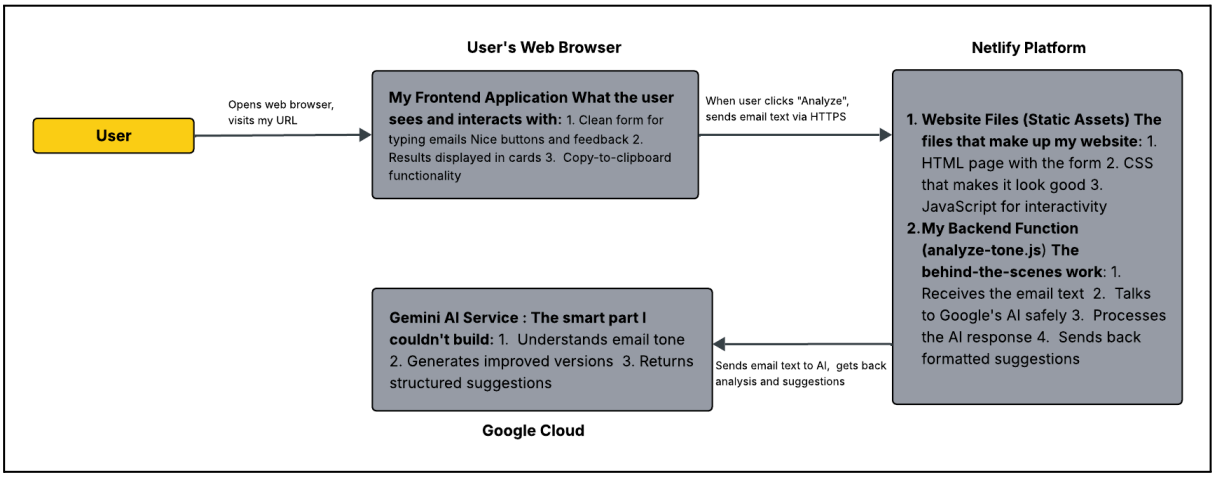

Fig 2. Container Diagram

3. **Level 3 : Component Diagram**

Breaking Down the User Interface : Once I knew I needed a frontend application, I had to figure out how to organize the code so it wouldn't become a mess. I broke it down into logical pieces that each handle one specific job.

Why I Organized It This Way : I learned from previous projects that mixing everything together makes bugs really hard to find and fix. So I separated concerns each component does one thing well, and they communicate through clear interfaces.

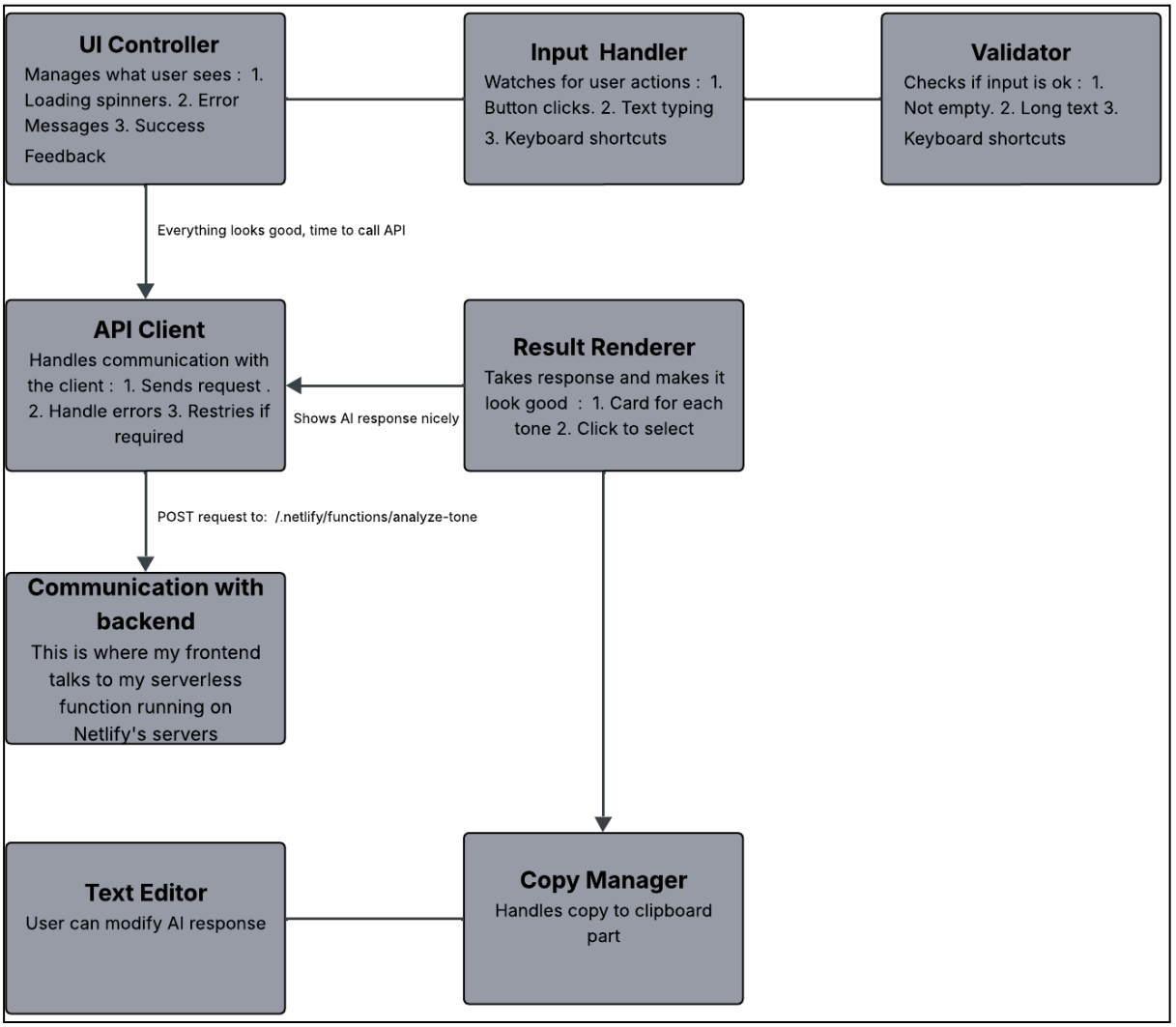

Fig 3. Component Diagram

4. **Code Diagram**

Level 4 (Code diagram) is not included as the application uses a simple functional JavaScript approach that doesn't require detailed class or module diagrams.