# Hello 👋

Thanks for taking a moment to look here.

I’m currently applying for a software engineering role at Apple, with a focus on systems, tooling, and testing infrastructure. This repository exists simply to provide a bit of additional context alongside my resume.

I’ve spent most of my career building reliable, testable systems — often in environments where correctness, determinism, and long-term maintainability mattered more than short-term velocity. That mindset is what draws me to Apple’s engineering culture.

I’ve worked professionally across all major operating systems:
- macOS and iOS
- Linux (server and containerized environments)
- Windows

That cross-platform experience has shaped how I think about tooling, abstractions, and developer experience — especially when building systems meant to be trusted by other engineers.

I care deeply about clear interfaces, thoughtful constraints, and software that behaves predictably under real-world conditions. I’m excited about the possibility of contributing to Apple’s frameworks and internal tools, and learning from the people who build them.

Thanks for reading,
**Travis**

---

## Engineering Notes

A few principles that have guided how I approach systems, tooling, and testing over the years:

- **Correctness before cleverness**
  I prefer simple, explicit designs that make invalid states hard or impossible to represent. When systems fail, they should fail loudly and predictably.

- **Testability is a design concern, not a phase**
  I’ve found that systems designed with clear boundaries and stable interfaces tend to be naturally testable, and require fewer brittle or overly-mocked tests.

- **Determinism builds trust**
  Whether generating documents, synchronizing state, or validating

data, I try to design systems that produce the same results given the same inputs. This makes failures easier to reason about and tools easier to trust.

- **Tooling should respect the developer**
  Internal tools and frameworks are successful when they reduce cognitive load, surface the right information at the right time, and stay out of the way when things are working.

- **Good abstractions age well**
  I value APIs and frameworks that are boring in the best way — stable, predictable, and resistant to accidental misuse.

These ideas have been shaped by working across backend systems, real-time applications, and multiple operating systems, and they continue to influence how I evaluate and build software today.