

Ziniu Zheng

Berkeley, CA | ziniuzheng@berkeley.edu | github.com/ZiniuZheng | linkedin.com/in/ziniu-zheng | ziniuzheng.github.io

EDUCATION

University of California, Berkeley

Berkeley, CA

Bachelor of Science, Electrical Engineering & Computer Sciences

Expected Dec 2027

Relevant Coursework: Data Structures, Machine Structures, Algorithms, Artificial Intelligence, Computer Graphics

TECHNICAL SKILLS

Programming Languages: Java, TypeScript, JavaScript, Go, Python, C

Databases and Cloud: PostgreSQL, MySQL, Elasticsearch, AWS (ECS, RDS, ECR), GCP (App Engine, GCS)

Web: React, Express, Spring Boot, Node.js, HTML/CSS

Tools & Libraries: Git, Docker, GitHub Actions, LangChain, Spring Security, JUnit, Mockito, Jest, React Testing Library, Ant Design, Material UI, Claude Code, Cursor

PROJECTS

ContextLens: AI Browser Assistant (Chrome Extension) | [Demo](#) *TypeScript, Chrome MV3, OpenAI/Anthropic*

- Built a Chrome extension that brings AI into any web page: select text to explain, summarize, or rewrite it in an in-place floating panel, with answers streaming token by token from OpenAI or Anthropic models
- Wrote an incremental Server-Sent Events parser that handles events split across network chunks; closing the panel disconnects the port and aborts the in-flight request
- Isolated the overlay in a closed Shadow DOM built with native DOM APIs and rendered model output as text only; kept API keys in the background service worker with host permissions limited to the two model APIs
- Shipped with zero runtime dependencies, 26 unit tests, and GitHub Actions CI (type-check, test, build) per push

Agent AI: Full-Stack RAG & MCP Document Q&A Agent | [Demo](#) *React, Node.js, LangChain, GPT-5, MCP*

- Built a document Q&A app that answers each question two ways, from an uploaded PDF and from live web search, with voice input and spoken answers (react-speech-recognition, speak-tts) and an Ant Design UI
- Implemented a RAG pipeline with LangChain: PDF loading, 500-character chunking, OpenAI embeddings and an in-memory vector store supplying retrieved context to GPT-5
- Built an MCP server exposing a SerpAPI search_web tool over stdio, and a GPT-5 tool-calling agent that writes its own search queries, using the document title to resolve references like “this paper”
- Traced intermittent timeouts to SerpAPI stalling on ~1/3 of uncached searches and added hedged requests; ran RAG and MCP in parallel with isolated failures, cutting answer time from ~15s to 6–12s

OnlineOrder: Online Food Ordering Web App | [Demo](#) *Spring Boot, PostgreSQL, React, Docker, AWS, JUnit*

- Developed RESTful APIs with Spring Boot covering registration, menu browsing, cart management, and checkout, structured into controller, service, and repository layers with dependency injection
- Used Spring Data JDBC with PostgreSQL on AWS RDS to manage restaurants, menus, and carts; added Caffeine caching with per-customer eviction, halving median menu API latency (~1.8× throughput) in local benchmarks
- Secured endpoints with Spring Security session-based authentication and BCrypt password hashing
- Wrote 27 JUnit 5 / Mockito unit tests for the service layer (ordering, checkout, sign-up), including Spring cache tests verifying @Cacheable/@CacheEvict invalidation
- Containerized the app with a multi-stage Docker build and set up GitHub Actions CI that runs backend and frontend tests and smoke-tests the image on every push
- Deployed to AWS ECS Fargate behind an HTTPS load balancer with a private RDS database; automated deployment and teardown with AWS CLI scripts

SocialAI: AI-Powered Social Network | [Demo](#) *Go, React, Elasticsearch, GCP, AWS Amplify, OpenAI*

- Built a full-stack social network with full-text post search: a Go REST API on Google App Engine, Elasticsearch on GCE (VPC-only), media in Google Cloud Storage, and a React frontend on AWS Amplify
- Moved OpenAI image generation behind a JWT-protected Go endpoint so the API key never reaches the browser, and migrated from the retired DALL-E 3 to the GPT Image API
- Fixed a signup race with atomic Elasticsearch creates: 10 concurrent signups → exactly 1 account
- Wrote 42 Go tests (incl. integration tests on real Elasticsearch) and 26 Jest tests, run in GitHub Actions CI