

# Production-Ready Web Development Projects: 27 GitHub Repositories Worth Studying

The best way to master modern web development is studying production-quality code. After comprehensive research across GitHub and live platforms, I've identified 27 exceptional open-source projects spanning 9 categories—from restaurant management systems to AI-powered resume builders. These aren't tutorial projects; they're battle-tested applications serving real users, with sophisticated architectures and enterprise-grade features. Together, they demonstrate the full spectrum of modern web development best practices, from real-time WebSocket communication to Progressive Web App implementations, from complex authentication systems to AI integration patterns.


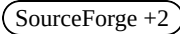
## Restaurant Business Website: Full-Stack Dining Platforms

Modern restaurant websites require far more than attractive templates—they need dynamic content management, reservation systems, and payment integration. The three projects below demonstrate production-ready architectures serving real restaurants.

### TastyIgniter: Battle-Tested Restaurant Management






**Repository:** <https://github.com/tastyigniter/TastyIgniter> (3,300+ stars)

**Live Demo:** <https://tastyigniter.com/demo>

TastyIgniter stands as the most mature option with **over 10 years of active development** and deployment in real restaurants worldwide.  Built on Laravel, it provides a complete restaurant management ecosystem with multi-location support, online ordering, and extensible plugin architecture.  +2

**Tech Stack:** Laravel (PHP 8.3+), MySQL/PostgreSQL, Laravel Blade templates, JavaScript 

**Notable Architecture:** The platform follows Laravel's MVC pattern with a sophisticated extension system allowing 20+ official plugins.  The modular design separates concerns between menu management, orders, reservations, and payment processing.  Event-driven architecture with webhooks enables integration with external services.

**What Makes It Worth Studying:** This is production-grade software with real-world complexity. Study it for comprehensive CMS integration patterns, multi-tenant architecture (managing multiple restaurant locations), payment gateway abstractions supporting Stripe/PayPal/Authorize.net,  and email template systems.  The codebase demonstrates professional Laravel practices including database migrations, seeding strategies, and service-oriented business logic layers.   Version 4.0.4 (September 2025) shows continued modernization. 

## Enatega Multi-Vendor Food Delivery System

**Repository:** <https://github.com/enatega/food-delivery-multivendor> (1,000+ stars)

**Live Demo:** <https://multivendor-admin.enatega.com/>

Enatega showcases a **GraphQL-based microservices architecture** with five separate applications: customer web app, customer mobile app (React Native), rider app, restaurant owner app, and admin dashboard. [GitHub](#)  
This demonstrates how to architect a complete ecosystem around a central API.

**Tech Stack:** React.js, React Native with Expo, Node.js/Express, MongoDB, GraphQL with Apollo, Firebase Authentication [github](#)

**Notable Architecture:** The entire system communicates through GraphQL subscriptions for real-time features like order tracking and rider location updates. Firebase handles authentication across all platforms (Google, Apple, Facebook), while Amplitude provides analytics and Sentry manages error tracking. [github](#) The frontend is fully open source, though the backend API requires a commercial license for production.

**What Makes It Worth Studying:** Examine this for **multi-platform state management** using Apollo Client and React Context, real-time GraphQL subscription patterns, push notification architecture with Firebase Cloud Messaging, and image optimization strategies using Imgix. The separation of five distinct apps sharing a common API demonstrates proper microservices boundaries. [github](#) The React Native implementation with Expo shows modern mobile development patterns including EAS Build configuration. [github](#)

## CosmicJS Next.js Restaurant with Headless CMS

**Repository:** <https://github.com/cosmicjs/nextjs-restaurant-website-cms> (~50 stars)

**One-Click Deploy:** Vercel-ready

This example demonstrates **Jamstack architecture** with Next.js and a headless CMS, showing how to separate content management from presentation. [GitHub](#) While smaller than the previous two, it exemplifies modern static site generation with dynamic content.

**Tech Stack:** Next.js 12+, Cosmic Headless CMS, Sass/SCSS, [Codementor](#) [DEV Community](#) Imgix for image optimization [GitHub +4](#)

**Notable Architecture:** Leverages Next.js Incremental Static Regeneration (ISR) to serve static pages that update when content changes. [Codementor](#) Cosmic CMS provides REST and GraphQL APIs with webhooks triggering rebuilds. [Codementor +3](#) The file-based routing and API routes demonstrate Next.js best practices for building full-stack applications.

**What Makes It Worth Studying:** This is your reference for **Jamstack patterns**, particularly the separation of content and code. Study the `getStaticProps` and `getServerSideProps` data fetching patterns, Next.js Image optimization techniques, environment variable management for API keys, and webhook integration for content-

triggered deployments. [Codementor +3](#) The SWC compiler usage (20-70x faster than Babel) shows performance optimization strategies. [Codementor](#) [DEV Community](#)

## Multiplayer Game Platform: Real-Time WebSocket Architecture

Building multiplayer games requires sophisticated networking, game state synchronization, and ranking systems. These three projects show different approaches to solving these challenges.

### Lichess: Elite-Scale Chess Platform

**Repository:** <https://github.com/lichess-org/lila> (15,000+ stars)

**Live Demo:** <https://lichess.org>

Lichess represents the **gold standard for online gaming platforms**, serving millions of players with sub-second latency. [GitHub](#) [Similarweb](#) Written in Scala, it demonstrates enterprise-scale WebSocket architecture and sophisticated matchmaking algorithms. [GitHub](#)

**Tech Stack:** Scala 3, Play Framework 2.8, TypeScript/Snabbdom frontend, MongoDB (4.7+ billion games), Redis pub/sub, Stockfish AI engine [github](#) [GitHub](#)

**Notable Architecture:** The system uses a **separate WebSocket server (lila-ws)** communicating with the main application via Redis pub/sub. [GitHub](#) This architectural separation allows horizontal scaling of WebSocket connections independent of game logic. Akka streams handle asynchronous processing throughout. [GitHub](#) The Glicko-2 rating system (evolution of ELO) calculates ratings for multiple time controls with mathematical rigor.

**What Makes It Worth Studying:** Study Lichess for **microservices at scale**—separate services handle HTTP, WebSockets, search (Elasticsearch), opening explorer, and tablebase lookups. [GitHub](#) The matchmaking algorithm balances rating ranges, time preferences, and wait times. Examine the cheat detection system using machine learning (Kaladin and Irwin tools), the tournament pairing algorithms supporting Swiss and Arena formats, and the distributed Stockfish cluster (fishnet) for computer analysis. [GitHub](#) The codebase shows pure functional programming patterns in Scala and proper separation between chess logic (scalachess module) and platform features.

### Cokepizza Chess: Production Node.js Implementation

**Repository:** <https://github.com/cokepizza/chess> (300+ stars)

**Live Demo:** <https://chesssup.com>

This project provides a **complete Socket.io chess platform** deployed on AWS, demonstrating practical patterns for building multiplayer games with Node.js. Unlike framework-based solutions, this shows every component implemented from scratch.

**Tech Stack:** Node.js, Express, React.js, Redux (Thunk & Saga), Socket.io, MongoDB with Mongoose, Express-Session [GitHub](#)

**Notable Architecture:** Socket.io namespaces divide functionality per page for optimization. The architecture sends only movement deltas (not entire board states) to minimize bandwidth. Redux manages complex client state with both Thunk (simple async) and Saga (complex flows) middleware. Session-based authentication shares state across browser tabs. (GitHub)

**What Makes It Worth Studying:** This demonstrates **custom chess algorithm implementation** without relying on external engines, showing how to implement piece logic, legal move validation, check/checkmate detection, and special moves (castling, en passant, promotion). The replay system with redo/undo uses notation-based board state jumping. (GitHub) Study the Socket.io optimization patterns—namespace division, partial board updates, and the HTTP request + Socket.io response pattern. The roadmap for Version 2 includes Redis integration for multi-process support, showing practical scaling considerations.

## boardgame.io: Universal Game Framework

**Repository:** <https://github.com/boardgameio/boardgame.io> (10,000+ stars)

**Live Demo:** <https://boardgame.io> with examples

Unlike complete platforms, boardgame.io is a **framework that abstracts away networking complexity**, letting developers focus on game logic. (Boardgame) It's used in commercial products and demonstrates excellent API design.

**Tech Stack:** JavaScript/TypeScript, React bindings, Node.js server, Socket.io for real-time, pluggable storage backends (npm)

**Notable Architecture:** Developers define games as pure functions describing state transitions. The framework handles all synchronization, networking, and storage automatically. (Boardgame) This separation of concerns—game logic vs infrastructure—shows excellent architectural abstraction.

**What Makes It Worth Studying:** Examine the **state management patterns** for turn-based games, including how the framework maintains consistency across clients and server. The built-in lobby API demonstrates matchmaking patterns. Study the automatically generated AI bots that work for any game, the game log system enabling time-travel debugging, and the plugin architecture for extending functionality. (GitHub +2) The framework supports secret state (hiding information from specific players), essential for card games. While it doesn't include ELO/ratings out of the box, the architecture makes adding them straightforward.

## Creative Tools Suite PWA: Offline-First Applications

Progressive Web Apps demonstrate the modern web's capabilities—installable, offline-functional, and performant. These creative tools showcase PWA best practices.

## Excalidraw: Production-Grade Collaborative Drawing

**Repository:** <https://github.com/excalidraw/excalidraw> (75,300+ stars, 208+ contributors)

**Live Demo:** <https://excalidraw.com>

Excalidraw is a **flagship PWA** demonstrating how the web platform matches native apps. Used by Google Cloud, Meta, Notion, and CodeSandbox, it proves PWAs are production-ready for professional tools. [GitHub +3](#)

**Tech Stack:** React, TypeScript, HTML5 Canvas API, IndexedDB, Service Workers, WebSocket for collaboration, End-to-End Encryption

**Notable Architecture:** The two-cache service worker strategy separates fonts (rarely change) from application resources (update frequently). [Web.dev](#) State management uses delta-based tracking—storing only changes rather than complete snapshots for efficient undo/redo. The File System Access API enables true open → edit → save workflows with OS integration.

**What Makes It Worth Studying:** This represents **PWA excellence**. Study the offline-first architecture where the app functions completely without network access, auto-saving to IndexedDB. The real-time collaboration uses WebSockets with E2E encryption (only participants decrypt content). [GitHub](#) [GitHub](#) Examine the Canvas rendering engine creating the hand-drawn aesthetic, the PNG/SVG export implementations, and the shape library system. The team explicitly chose PWA over Electron, citing better web capabilities—a decision worth understanding. [Excalidraw+](#) [Web.dev](#) The npm package ([@excalidraw/excalidraw](#)) shows how to design embeddable components. [npm](#)

## Scribbles: Multilayer Drawing PWA

**Repository:** <https://github.com/JuanLPalacios/Scribbles> (5 stars, actively developed)

**Live Demo:** Installable via app's Quick Start modal

Scribbles demonstrates **sophisticated brush systems** with pressure sensitivity, texture brushes, pattern brushes, and multilayer support—features typically requiring native apps. [GitHub](#)

**Tech Stack:** React, TypeScript, HTML5 Canvas, IndexedDB, Service Workers, JSZip for custom formats, ABR-JS for Adobe Brush support

**Notable Architecture:** The custom .scribble format uses zip-based compression storing multiple layers. The brush engine implements pressure sensitivity via pointer events. Layer compositing uses Canvas blend modes and opacity controls.

**What Makes It Worth Studying:** Study the **custom brush implementation**—how to handle texture brushes, pattern repetition, and pressure curves. The mobile-first touch interface shows how to optimize Canvas interactions for touch. The Adobe Brush format support (ABR-JS) demonstrates integration with industry standards. Examine the zero-backend approach—all features work purely offline. The installability pattern lets users install directly from the app interface via service worker prompts.

## tldraw: Collaborative Infinite Canvas SDK

**Repository:** <https://github.com/tldraw/tldraw> (41,700+ stars)

**Live Demo:** <https://tldraw.com>

While primarily an SDK rather than standalone PWA, tldraw demonstrates **high-performance infinite canvas** architecture and real-time collaboration patterns applicable to any creative tool. [GitHub +2](#)

**Tech Stack:** React, TypeScript SDK, [GitHub](#) [npm](#) Cloudflare Durable Objects for sync (optional), localStorage persistence, WebGL for advanced rendering

**Notable Architecture:** The powerful Editor API (god object pattern) centralizes all canvas operations. [tldraw](#) Real-time sync via @tldraw/sync library uses Cloudflare Durable Objects, demonstrating edge computing for low latency. [tldraw](#) [tldraw](#) Custom shape support uses React components for rendering. [tldraw](#)

**What Makes It Worth Studying:** Examine the **infinite canvas mathematics**—viewport transformations, zooming, panning, and coordinate system management. The localStorage persistence using the persistenceKey prop shows simple offline patterns. [tldraw](#) Study the gesture recognition system handling mouse, touch, and pen inputs uniformly. The WebGL shader integration capability (shader starter kit) demonstrates advanced rendering techniques. [GitHub](#) Note the commercial licensing model (watermark removal or paid license)—an important consideration for production use. [GitHub +2](#)

## SaaS Admin Platform: Enterprise Authentication and RBAC

Complete SaaS platforms require sophisticated authentication, authorization, real-time features, and data visualization. These three projects demonstrate production-ready patterns.

### Wasp Open SaaS: Configuration-Driven Full-Stack

**Repository:** <https://github.com/wasp-lang/open-saas> (~10,000+ stars)

**Live Demo:** <https://opensaas.sh>

Wasp Open SaaS uses a **declarative configuration approach** where you define features in a config file and the framework generates full-stack implementation. [GitHub](#) This reduces boilerplate by approximately 70%.

**Tech Stack:** React 18 with TypeScript, Node.js/Express, Wasp framework, Prisma ORM (PostgreSQL), Tailwind + ShadCN UI, Astro for landing pages [github](#)

**Notable Architecture:** The Wasp framework handles routing, authentication, database operations, and background jobs from simple declarations. [GitHub](#) For example, auth configuration in a few lines generates complete email verification, OAuth flows, and session management. [Wasp](#) End-to-end type safety works without GraphQL or tRPC—the framework generates typed API clients. [github](#)

**What Makes It Worth Studying:** This demonstrates **reducing accidental complexity**. Study how declarative configuration generates imperative code, the patterns for background jobs (cron tasks, queues) via Wasp's Jobs

feature, and deployment simplification (one-command deploy to Railway, Fly.io). (GitHub) The integration patterns for Stripe payments, OpenAI API, SendGrid/MailGun emails show clean service abstractions. (Wasp) Examine the RBAC implementation and how role checks integrate throughout the stack. The Playwright E2E tests demonstrate testing strategies. (github)

## async-labs/saas: Battle-Tested Production SaaS

**Repository:** <https://github.com/async-labs/saas> (4,200+ stars)

**Live Demo:** <https://saas-app.async-await.com>

This project powers **multiple real products** (Work in Biotech, Async, AI-cruiter), proving its production readiness. (Medium) (GitHub) The codebase shows patterns learned from operating actual SaaS businesses.

**Tech Stack:** React 18/TypeScript, Next.js (SSR), Node.js/Express, MongoDB (Mongoose), Material-UI, MobX state management, Socket.io v3, AWS S3/SES

**Notable Architecture:** The **monorepo separates app (Next.js) and api (Express)** servers—the app server handles SSR and client routing while the API server processes business logic. This separation allows independent scaling. Socket.io enables real-time features including discussion threads, notifications, and team collaboration. Team-based RBAC with Owner/Admin/Member roles demonstrates multi-tenant patterns.

**What Makes It Worth Studying:** Study the **team-based multi-tenancy** architecture with proper data isolation using MongoDB queries. The authentication with Google OAuth and session management shows production security patterns. (GitHub) Examine the Socket.io integration for real-time updates, the Material-UI SSR configuration (tricky with styling), and the MobX state management patterns. The AWS integration demonstrates file uploads (S3 pre-signed URLs), transactional emails (SES), and deployment (Elastic Beanstalk). Row-level security patterns ensure users only access authorized data.

## ixartz/SaaS-Boilerplate: Modern Next.js with Enterprise Auth

**Repository:** <https://github.com/ixartz/SaaS-Boilerplate> (substantial community following)

**Documentation:** Comprehensive README

This boilerplate showcases **Next.js 14 App Router** with modern patterns including Server Components, demonstrating cutting-edge React features in production contexts.

**Tech Stack:** Next.js 14+ (App Router), React 18, TypeScript, Drizzle ORM (type-safe SQL), PostgreSQL/MySQL/SQLite, Clerk authentication, Shadcn UI, Sentry, Better Stack logging

**Notable Architecture:** Clerk provides enterprise authentication including **built-in 2FA/MFA** without custom implementation. (GitHub) Drizzle ORM offers compile-time SQL type safety—typos in queries cause TypeScript errors. The Next.js 14 App Router uses React Server Components for better performance.

**What Makes It Worth Studying:** This represents **modern Next.js best practices**. Study the Server Components vs Client Components patterns, the Drizzle ORM migration system with rollback support, and how Clerk handles authentication (social login, magic links, passkeys, MFA). Examine the multi-tenancy via organizations/teams, the internationalization setup (next-intl + Crowdin), and the comprehensive testing (Vitest for units, Playwright for E2E, Codecov for coverage). The Sentry integration with Spotlight (dev mode debugging) demonstrates production error handling. Security headers achieve A+ rating—review the configuration.


## Personal Productivity Hub: Cross-Platform Sync

Building productivity apps requires integrating multiple features with seamless sync across devices. These projects show different architectural approaches.



### Super Productivity: Comprehensive Production App



**Repository:** <https://github.com/johannesjo/super-productivity> (14,500+ stars)

**Live Demo:** <https://app.super-productivity.com/>

Super Productivity is a **complete productivity suite** with task management, goal tracking, habit tracking, and Pomodoro timer, all integrated into a polished application serving thousands of users.  +2

**Tech Stack:** Angular 18, TypeScript, RxJS, Electron (Windows/macOS/Linux), Capacitor for Android, IndexedDB for local storage


**Notable Architecture:** The privacy-first, local-first design stores all data in IndexedDB with optional sync via WebDAV or Dropbox—no central backend.   The RxJS reactive architecture handles complex state flows. Capacitor bridges the web codebase to native Android capabilities.

**What Makes It Worth Studying:** This demonstrates **feature integration at scale**. Study how multiple productivity features (tasks, goals, habits, timer) share common data models and sync infrastructure. The anti-procrastination features using CBT (Cognitive Behavioral Therapy) techniques show thoughtful UX design. Examine the integration patterns with external services (Jira, GitHub, GitLab, Gitea, OpenProject, Redmine, YouTrack, CalDAV)—APIs are abstracted cleanly.  +2 The sync conflict resolution strategies for WebDAV/Dropbox handle offline editing. The Electron packaging and auto-update mechanisms demonstrate desktop app distribution. 

### Flow Hypermind: Firebase-Powered Mobile App

**Repository:** <https://github.com/Lordhacker756/Flow> (~30 stars, moderate activity)

**Platform:** React Native (iOS/Android)

Flow demonstrates **Firebase as a complete backend**, showing how to build a full-stack mobile app using Firebase services for authentication, database, and potential cloud functions. 



**Tech Stack:** React Native CLI, Firebase (Authentication, Firestore), React Navigation, Redux Toolkit, React Native Charts

**Notable Architecture:** Firebase Firestore provides real-time sync automatically—listeners update components when data changes. Redux Toolkit manages global state with modern patterns (slices, thunks). Multi-method authentication (Email, Google, Phone OTP) uses Firebase Auth. ([GitHub](#))

**What Makes It Worth Studying:** Study **Firebase integration patterns**—authentication flows, Firestore data modeling (collections/subcollections), offline persistence enabling offline-first behavior, and real-time listeners for live data. The React Native Charts integration (Doughnut, Line charts) shows data visualization on mobile. ([github](#)) Examine the navigation setup combining Stack and Tab navigators. While incomplete (Redux implementation partial, notifications planned), it demonstrates clean React Native architecture and Firebase patterns applicable to many mobile projects.

## Habitica Lite: Microservices Reference Architecture

**Repository:** <https://github.com/haxejs/habitica-lite> (~30 stars, archived/legacy)

**Status:** Proof of concept, not actively maintained

Though archived, Habitica Lite demonstrates **microservices architecture** with job scheduling, real-time sync, and multi-platform clients (web, mobile, WeChat).

**Tech Stack:** Node.js with Strongloop (Express-based), MongoDB, Socket.io, Angular 2/4 (web), NativeScript + Angular (iOS/Android), Agenda for job scheduling ([github](#))

**Notable Architecture:** Strongloop provides a robust REST API framework with built-in API explorer (/explorer endpoint). Agenda handles background jobs including daily cron tasks and reminder notifications. Socket.io enables real-time bidirectional communication. ([github](#)) Docker configuration demonstrates deployment patterns. ([github](#))

**What Makes It Worth Studying:** Despite being legacy (Node v6, Angular 2), the **architectural patterns remain valuable**. Study the microservices boundaries, the job scheduling system for background tasks (Agenda), the Socket.io real-time patterns, and the shared codebase between web (Angular) and mobile (NativeScript + Angular). The Docker deployment setup demonstrates containerization. Note this would require significant updates to be production-ready—treat it as an architectural reference rather than a starting point.

## Classic Games Hub with AI: Algorithms and Platforms

Implementing game AI and managing multiple games requires sophisticated algorithms and clean architecture. These projects demonstrate different approaches.

## Lichess: AI and Platform Excellence

(See full details in Multiplayer Game Platform section above)

**Additional AI Focus:** Lichess integrates the **Stockfish engine** (world's strongest chess engine) via a distributed cluster called fishnet. (GitHub) Volunteer computers donate processing power for analysis. The system demonstrates how to integrate external AI engines, distribute computational load, and provide multiple AI difficulty levels. The opening book and endgame tablebase integration shows knowledge base architecture.

## TabletopGames Framework (TAG): Academic AI Research

**Repository:** <https://github.com/GAIGResearch/TabletopGames> (200+ stars)

**Website:** <http://tabletopgames.ai>

TAG is a **Java-based framework from Queen Mary University** featuring 17+ implemented games including Tic-Tac-Toe, Connect 4, Catan, Pandemic, Terraforming Mars, Dominion, and more. It's designed for AI research.

**Tech Stack:** Java 8+, Maven, JSON configuration (github)

**Notable Architecture:** The common API for AI agents allows implementing one algorithm (minimax, MCTS) that works across all games. Reusable component system (cards, dice, boards, tokens) speeds up game implementation. (github) Game analytics track action space, branching factor, and hidden information automatically.

**What Makes It Worth Studying:** This demonstrates **clean game abstraction**. Study how the component system generalizes across vastly different games (board games, card games, strategy games), the AI agent API design allowing algorithm reuse, and the analytics framework measuring game complexity. The minimax implementation with alpha-beta pruning shows classic game AI. Monte Carlo Tree Search (MCTS) demonstrates modern AI approaches for games with large state spaces. The published academic paper (EXAG 2020) provides theoretical grounding. Note this is primarily for AI research—adding WebSocket multiplayer and tournament systems would be custom implementation.

## boardgame.io: Framework for Custom AI

(See full details in Multiplayer Game Platform section above)

**Additional AI Focus:** The framework **automatically generates AI bots** for any game by simulating moves and evaluating outcomes. (GitHub +2) For sophisticated games, developers can implement custom Monte Carlo Tree Search (MCTS) algorithms. The architecture separates game state from AI decision-making cleanly. Study how the automatically generated bots work through random play simulation and how to override with custom strategies.

## Algorithm Visualizer Platform: Educational Visualization

Interactive algorithm visualizations help students understand complex concepts. These platforms demonstrate educational tool development with sophisticated animations.

## Algorithm Visualizer: Interactive Code Execution

**Repository:** <https://github.com/algorithm-visualizer/algorithm-visualizer> (47,800+ stars)

**Live Demo:** <https://algorithm-visualizer.org>

This platform's unique approach lets users **write algorithms and visualize them executing**. Custom tracer libraries extract visualization commands from actual code in JavaScript, C++, Java, or Python. [GitHub](#)

[Best of JS](#)

**Tech Stack:** React, Node.js backend, custom tracer libraries (JavaScript/C++/Java/Python), GitHub authentication [GitHub](#) [Best of JS](#)

**Notable Architecture:** The server safely compiles and executes user code, extracting tracer commands embedded in comments. The visualization panel synchronizes with code editor highlighting. Microservices architecture separates web app, server, algorithm repository, and tracer libraries. [GitHub](#) [Best of JS](#)

**What Makes It Worth Studying:** Study the **code execution visualization pipeline**—how user code is parsed, executed server-side for security, and tracer commands are sent to the frontend. The multi-language tracer libraries show how to create polyglot systems. Examine the synchronization between code highlighting and visual state changes, the Canvas-based rendering for algorithm animations, and the GitHub integration for saving work. [GitHub](#) [Best of JS](#) The comprehensive algorithm library covering sorting, searching, graphs, trees, DP, divide-and-conquer, and backtracking demonstrates educational content organization.

## TamimEhsan's AlgorithmVisualizer: Modern Next.js Implementation

**Repository:** <https://github.com/TamimEhsan/AlgorithmVisualizer> (297 stars, recently updated to v2.0.0)

**Live Demo:** <https://tamimehsan.github.io/AlgorithmVisualizer/>

Recently migrated to **Next.js 14 with ShadCN UI**, this visualizer covers 24+ algorithms across 8 segments including unique features like Turing machine visualization and recursion tree visualization. [github](#) [Tamimehsan](#)

**Tech Stack:** Next.js 14+, ShadCN UI components, Tailwind CSS, custom Canvas animations [github](#)

**Notable Architecture:** Component-based React architecture with clean separation between visualization logic and UI. Custom CSS animations without heavy libraries maintain performance. Multiple visualization modes—grids for pathfinding, bar charts for sorting, trees for recursion.

**What Makes It Worth Studying:** This demonstrates **diverse visualization techniques**. Study the pathfinding visualizations (DFS, BFS, Dijkstra, A\*) on interactive grids where users create custom mazes. The recursion tree visualizer for Fibonacci, binomial coefficients, and derangements shows how to visualize recursive calls. The computational geometry implementation (Graham Scan convex hull) demonstrates geometric algorithm animation. The Turing machine visualizer for bitwise operations provides unique educational value. [github](#)

[Tamimehsan](#) The migration to Next.js 14 and ShadCN UI demonstrates modernization patterns for legacy projects.

## algo-visualizers: Performance-Optimized with Metrics

**Repository:** <https://github.com/sadanandpai/algo-visualizers> (711 stars, v2.0.0 October 2024)

**Live Demo:** <https://sadanandpai.github.io/algo-visualizers/>

This visualizer uses **pure CSS animations with the FLIP principle** (First, Last, Invert, Play) for maximum performance, avoiding heavy animation libraries while achieving smooth animations. [github](#)

**Tech Stack:** React 18, Redux Toolkit, Vite, Sass (SCSS), Vitest + Cypress testing, Lucide icons, React Joyride tours [github +2](#)

**Notable Architecture:** CSS Flexbox order property enables efficient element reordering without DOM manipulation. JavaScript async generators control algorithm execution step-by-step. Redux Persist maintains user preferences. [github](#) Production-grade testing with both unit (Vitest) and E2E (Cypress) tests.

**What Makes It Worth Studying:** Study the **FLIP animation technique**—calculate starting position (First), calculate ending position (Last), invert to create illusion of smooth transition (Invert), play the animation (Play). This pure CSS approach outperforms JavaScript animation libraries. Examine the metrics display showing swaps, comparisons, and time—essential for understanding algorithm efficiency. The side-by-side algorithm comparison feature demonstrates how to run multiple visualizations simultaneously. The async generator pattern controls algorithm stepping, allowing play/pause/speed controls. [github](#) The comprehensive testing setup (unit + E2E) ensures reliability.

## Professional Resume Builder SaaS: Document Generation

Resume builders require sophisticated PDF generation, template systems, and increasingly, AI assistance. These three projects demonstrate production patterns.

### Reactive Resume: Feature-Complete SaaS

**Repository:** <https://github.com/AmruthPillai/Reactive-Resume> (20,000+ stars)

**Live Demo:** <https://rxresu.me>

Reactive Resume is the **most comprehensive open-source resume builder**, featuring 12+ templates, OpenAI integration, version control, and complete SaaS features including authentication and public resume links with analytics. [GitHub](#) [github](#)

**Tech Stack:** React with TypeScript, Next.js (App Router), NestJS backend, PostgreSQL (Prisma), Minio object storage, Browserless (headless Chrome), Redis caching [GitHub](#)

**Notable Architecture:** The NestJS backend provides a robust API with authentication, authorization, and business logic. Browserless uses headless Chrome for high-quality PDF rendering with embedded fonts ensuring consistent cross-platform rendering. Minio provides S3-compatible object storage for avatars and PDFs. [GitHub](#) Row-level security ensures data isolation.

**What Makes It Worth Studying:** This demonstrates **complete SaaS architecture**. Study the authentication system (GitHub, Google, Email + 2FA), the multi-resume version control system (create base resumes and tailored versions), and the public resume link feature with analytics (track views/downloads). The OpenAI integration uses BYOK (Bring Your Own Key) stored locally for privacy—API calls go directly from browser to OpenAI. The drag-and-drop section customization shows advanced UI patterns. [GitHub](#) [GitHub](#) Examine the PDF generation pipeline using headless Chrome, the template system architecture supporting 12+ designs, and the import features (existing PDFs, LinkedIn data). [Reactive Resume](#) The monorepo structure and Docker Compose deployment demonstrate microservices patterns.

## OpenResume: Privacy-First with ATS Testing

**Repository:** <https://github.com/xitanggg/open-resume> (5,500+ stars)

**Live Demo:** <https://open-resume.com> [Open Source Agenda](#)

OpenResume is **100% client-side** with no backend, no sign-up, and no data collection. Its sophisticated PDF parser tests ATS (Applicant Tracking System) readability—a unique educational feature. [github +2](#)

**Tech Stack:** React with TypeScript, Next.js 13 (App Router), Redux Toolkit, Tailwind CSS, react-pdf for generation, PDF.js (Mozilla) for parsing [github](#) [GitHub](#)

**Notable Architecture:** Entirely client-side—runs in browser with all data stored locally. No authentication needed, works offline after initial load. The PDF generation uses react-pdf (@react-pdf/renderer) rendering PDFs purely in JavaScript. [github](#) [GitHub](#)

**What Makes It Worth Studying:** Study the **ATS parser implementation**—a sophisticated four-step algorithm: (1) PDF.js reads PDF into text items, (2) groups text items into lines with context, (3) groups lines into sections (Education, Experience, Skills), (4) feature scoring system extracts 100+ attributes. The parser uses heuristics to detect names, emails, phone numbers, and section structures, demonstrating practical NLP-lite techniques. [OpenResume](#) Examine the client-side architecture patterns making the app work without a backend, the Redux Toolkit state management, and the react-pdf rendering. The single professional template is optimized for U.S. standards and ATS compatibility—study the design decisions (no photos, single-column layout, semantic structure).

## ResumeLM: Multi-AI Integration

**Repository:** <https://github.com/olyaiy/resume-lm> (actively maintained)

**Tech Stack:** React 19, Next.js 15, TypeScript, Supabase, PostgreSQL, Shadcn UI, Framer Motion, NextAuth [github](#)

ResumeLM demonstrates **cutting-edge AI integration** supporting five AI providers: OpenAI GPT, Claude (Anthropic), Google Gemini, DeepSeek, and Groq. This multi-provider approach offers flexibility and redundancy. [GitHub](#) [github](#)

**Notable Architecture:** Supabase provides backend-as-a-service with authentication, database, and row-level security. The Next.js 15 App Router uses React 19 Server Components. NextAuth handles authentication flows. The database schema includes sophisticated features: user\_profiles with JSONB for flexible data, resumes with section\_order and configs, jobs table with requirements, applications table linking resumes to jobs. (github)

**What Makes It Worth Studying:** This showcases **advanced AI integration patterns**. Study how to abstract multiple AI providers behind a common interface, making it easy to switch providers or use multiple simultaneously. The AI-powered features include bullet point improvement (claimed 90% more effective), content suggestions based on experience, keyword optimization, and job description tailoring. (GitHub) (github) Examine the job tracking system integrating with resume versions, the version control implementation, and the performance metrics (detailed analytics). The JSONB usage for flexible salary ranges and complex configurations demonstrates PostgreSQL advanced features. The Framer Motion integration shows smooth animations enhancing UX. The Stripe integration demonstrates payment processing patterns.

## Community Forum Platform: Complex Social Software

Forum platforms require sophisticated features including threading, voting, moderation, search, and real-time updates. These three production platforms demonstrate different architectural approaches.

### Discourse: Ruby Excellence at Scale

**Repository:** <https://github.com/discourse/discourse> (45,300+ stars)

**Live Demo:** <https://try.discourse.org> (resets daily) and <https://meta.discourse.org>

Discourse is the **most mature open-source forum platform**, powering thousands of communities. (GitHub) Built with Ruby on Rails and Ember.js, it demonstrates how traditional technologies can create modern, interactive experiences.

**Tech Stack:** Ruby on Rails (RESTful JSON API), Ember.js (SPA), PostgreSQL 13+, Redis 7+ (jobs/cache/real-time)

**Notable Architecture:** Rails provides a robust API backend while Ember.js creates a responsive single-page application. Redis handles multiple concerns—job queues (Sidekiq), caching, rate limiting, and transient data. The trust level system (cornerstone feature) sandboxes new users and gradually grants privileges as they contribute positively.

**What Makes It Worth Studying:** Study the **trust level implementation**—an automated reputation system that prevents spam while rewarding good contributors. The voting system uses "likes" rather than up/down votes, creating positive community dynamics. Examine the granular permission system with role-based access per category, the moderation queue with flagging/review tools, and the automatic anti-spam with Akismet. The plugin architecture enables extensibility—study the Extension API design. Real-time features use WebSockets

for live updates and streaming discussions. The built-in chat functionality and ActivityPub support (fediverse integration) show forward-thinking features. GDPR compliance by design demonstrates privacy considerations.

## **NodeBB: Modern Node.js Forum**

**Repository:** <https://github.com/NodeBB/NodeBB> (14,700+ stars)

**Live Demo:** <https://try.nodebb.org> and <https://community.nodebb.org>

NodeBB demonstrates **WebSocket-first architecture** with Node.js, offering real-time experiences out of the box. Its unique database flexibility supports MongoDB, Redis, or PostgreSQL.

**Tech Stack:** Node.js 20+ (Express.js), WebSockets for real-time, MongoDB/Redis/PostgreSQL (choose one), Bootstrap 5-based themes

**Notable Architecture:** The WebSocket-first design makes all interactions feel instant—posts appear without page refreshes, notifications arrive immediately. The flexible database backend (MongoDB, Redis, or PostgreSQL) lets you choose based on requirements. When clustering, Redis is required for WebSocket synchronization across processes.

**What Makes It Worth Studying:** Study the **multi-database abstraction layer**—how to write code supporting different database paradigms (document, key-value, relational). The WebSocket implementation demonstrates real-time patterns including chat, live feeds, and instant notifications. Examine the plugin system (100+ community plugins) with examples ranging from authentication to SEO to analytics. The theme engine shows how to create customizable UIs with Bootstrap 5 as a base. The Q&A plugin provides Stack Overflow-style functionality, demonstrating plugin extensibility. The scheduled topics feature and social media sharing integration show thoughtful UX additions.

## **Apache Answer: Q&A Platform for Knowledge Bases**

**Repository:** <https://github.com/apache/answer> (7,900+ stars, Apache Incubator)

**Live Demo:** <https://meta.answer.dev>

Apache Answer provides a **Stack Overflow alternative** focused on Q&A workflows. As an Apache Incubator project, it demonstrates professional open-source governance.

**Tech Stack:** Go (Golang backend), React with TypeScript (frontend), MySQL or PostgreSQL, WebSockets, Docker-first deployment

**Notable Architecture:** Go provides performance and concurrency advantages. React with TypeScript creates a type-safe frontend. The plugin architecture includes Connector, Storage, Cache, Search, and Notification plugins—clean separation of concerns.

**What Makes It Worth Studying:** Study the **Q&A-specific features**—question/answer voting, accepted answer workflow (question owner marks best answer), and the collaborative editing system where anyone can

edit questions/answers with full revision history. The reputation system unlocks privileges at thresholds, demonstrating gamification principles. Examine the search implementation with Elasticsearch and Meilisearch plugin support, the tag system for content organization, and the follow tags feature for personalized feeds. The Markdown editor with real-time preview shows modern content creation UX. The @mention system invites experts to questions. The revision history tracking every edit demonstrates transparency principles.

## Synthesis: Key Patterns Across All Projects

After analyzing 27 repositories, several patterns emerge that distinguish production-quality code from tutorials.

**Authentication Excellence:** The best projects use established providers (Clerk, Firebase, NextAuth, OAuth 2.0) rather than rolling custom auth. They implement 2FA/MFA, session management, and proper password handling. Study Wasp Open SaaS for declarative auth config, ixartz/SaaS-Boilerplate for Clerk integration, and async-labs/saas for Google OAuth patterns.

**Real-Time Architecture:** WebSocket implementation varies—Lichess uses separate WebSocket servers with Redis pub/sub for scale, NodeBB uses WebSockets natively in Node.js, Discourse uses Rails channels. The pattern: separate real-time concerns from business logic, use Redis for multi-process coordination, and send minimal data (deltas, not full states).

**Database Strategy:** Modern projects favor PostgreSQL for relational needs (type safety, ACID guarantees) and MongoDB for flexible schemas. Prisma and Drizzle ORMs provide type safety. Super Productivity and OpenResume demonstrate IndexedDB for client-side storage. The key: match database to access patterns.

**Testing Approaches:** Production projects include comprehensive testing—Vitest for units, Playwright/Cypress for E2E, Codecov for coverage tracking. Study ixartz/SaaS-Boilerplate and algo-visualizers for complete testing setups. The pattern: test critical paths thoroughly, use E2E for user flows, achieve high coverage on business logic.

**AI Integration Patterns:** Three approaches emerge: (1) BYOK (Bring Your Own Key) where users provide API keys stored client-side (Reactive Resume), (2) Server-side proxying where your backend calls AI services (ResumeLM's multi-provider approach), (3) Direct client calls with exposed keys (suitable for free tiers). Choose based on cost structure and privacy needs.

**Deployment Strategies:** Docker dominates for backend services, Vercel for Next.js, and various hosting for specialized needs. Study Wasp Open SaaS for one-command deployment, Discourse for traditional server deployment, and OpenResume for static hosting. The trend: containerization for consistency, platform-as-a-service for speed.

## Choosing Your Learning Path

Start with projects matching your experience level. **Beginners** should study OpenResume (client-only



simplicity), TamimEhsan's AlgorithmVisualizer (clean React patterns), or CosmicJS Next.js (modern Jamstack).

**Intermediate developers** benefit from Super Productivity (complex state management), boardgame.io (excellent API design), or Reactive Resume (complete SaaS patterns). **Advanced developers** should analyze Lichess (scale and microservices), async-labs/saas (production battle-testing), or Discourse (mature architecture).

For **specific skills**, target accordingly: Real-time features (Lichess, NodeBB, cokepizza/chess), PWA implementation (Excalidraw, Scribbles), AI integration (ResumeLM, Reactive Resume), Authentication (Wasp Open SaaS, ixartz/SaaS-Boilerplate), or Algorithm visualization (Algorithm Visualizer, algo-visualizers).

These 27 repositories represent thousands of hours of production experience. Study their architectures, examine their patterns, and apply their lessons to build your own exceptional web applications.