Rules.mdc

```
---
description: Clean Code Implementation and Project Management Best Practices
globs:
 - "**/*.ts"
 - "**/*.js"
 - "**/*.tsx"
 - "**/*.jsx"
alwaysApply: true
---

# Clean Code Implementation Techniques

- Use descriptive, intention-revealing names for variables, functions, classes, etc.
- Avoid vague or misleading names to improve readability and maintainability.
- Each function or module should have one clear purpose, following Single Responsibility Principle (SRP).
- Write code that clearly expresses intent to minimize comments; comment "why", not "what".
- Replace magic numbers or strings with named constants for clarity.
- Organize code into layers or modules (routes, controllers, services, models).
- Implement centralized and consistent error handling.
- Use modern language features like async/await for better async operations management.
- Use code linting and formatting tools (ESLint, Prettier) automatically.
- Write unit tests to ensure correctness and ease future refactoring.
- Avoid duplications by abstracting repeated logic into reusable functions/classes.
- Enforce coding standards using linters and pre-commit hooks.
- Regularly refactor code for simplicity and reduced technical debt.

# Project Management and Collaboration
```

- Define clear project scope, objectives, deliverables, deadlines, and constraints.
- Assign clear roles and responsibilities for team members.
- Use project management tools for task, version, and documentation tracking.
- Practice regular communication, standups, reviews, and retrospectives.
- Design APIs/modules to be idempotent and implement caching/memoization.
- Use code reviews with multiple reviewers and integrate automated checks.
- Employ branching strategies like GitFlow and commit with descriptive messages.
- Maintain detailed project documentation, including API docs and architecture decisions.
- Automate repetitive tasks such as builds, deployments, and code quality checks.
- Use effective communication tools (Slack, Teams) for streamlined interactions.
- Share reusable code snippets consistently.
- Keep audit trails with reasons and author for changes and enforce access controls.
- Adapt conflict resolution styles and encourage collaborative problem-solving.

.claude.md

Formatting

1. Analysis & Planning

- Define the problem statement clearly.
- Identify key objectives and success criteria.
- Outline project scope and constraints.
- Determine data sources, tools, and technologies required.
- Develop a high-level roadmap and timeline.
- Identify stakeholders and communication plan.

2. Components Research Phase

- Investigate relevant algorithms, models, and frameworks.
- Evaluate existing libraries and APIs.
- Research domain-specific requirements and best practices.
- Analyze potential technical challenges and risks.
- Document findings and justify component selections.

3. Implementation code Phase

- Set up development environment and dependencies.
- Write modular, reusable, and testable code.
- Implement core functionalities first, then auxiliary features.
- Integrate error handling and logging.
- Continuously test and validate each component.
- Maintain clear and consistent documentation within the code.

4. Apply themes

- Ensure modularity for ease of updates and scaling.
- Apply security best practices (e.g., input validation, auth).
- Focus on maintainability and extensibility.
- Enhance usability and user experience where applicable.
- Follow ethical AI guidelines including fairness and transparency.

Design Principles

- Single Responsibility Principle for each module.
- KISS (Keep It Simple, Stupid) for clarity.
- DRY (Don’t Repeat Yourself) to reduce redundancy.
- Scalability to handle growing data or users.
- Robustness to recover from errors gracefully.

Code Quality Standards

- Use consistent naming conventions.
- Follow language-specific style guides and linters.
- Include comments and docstrings for complex logic.
- Write unit and integration tests with good coverage.
- Perform code reviews and incorporate feedback.
- Use version control (e.g., Git) with meaningful commits.

Integration Guidelines

- Define clear API contracts and data formats.
- Ensure backward compatibility where possible.

- Validate inputs and outputs rigorously.

- Establish communication protocols (e.g., REST, gRPC).

- Use middleware to handle cross-cutting concerns (e.g., auth, logging).

- Document integration points for other developers.

Performance Optimization

- Profile code to identify bottlenecks.

- Use efficient data structures and algorithms.

- Cache expensive computations when appropriate.

- Employ asynchronous processing where beneficial.

- Minimize memory footprint and network calls.

- Continuously monitor and tune in production.

Output format

- Define output data structure (JSON, XML, CSV, etc.).

- Ensure outputs contain necessary metadata and timestamps.

- Format data for human-readability where needed.

- Include error messages with clear codes and descriptions.

- Provide standardized logs for auditing and troubleshooting.

Rules

- Adhere to data privacy laws and regulations (GDPR, CCPA, etc.).

- Maintain ethical standards in AI model usage and data handling.

- Enforce security policies on data access and storage.

- Limit external dependencies to trusted sources.

- Ensure reproducibility of results across environments.

- Regularly update dependencies to patch vulnerabilities.

Project Planning Formatting

Project Foundation

Project Name

Description:

Problem Statement

Description:

Solution Overview

Description:

Include:

Core technology/approach

Key features

How it's different from existing solutions

Expected impact

Sections:

Technical Stack: List all technologies, frameworks, and tools

Architecture: Explain system design and data flow

User Stories: Describe how users will interact with your solution

Technical Challenges: Highlight innovative solutions to complex problems

Scalability: Explain how your solution can grow

Community Engagement Features

Description: Features that encourage user interaction, feedback, and community building around your project.

Examples:

User feedback systems

Community forums or chat

Gamification elements

Collaborative features

Open source contributions

Design & User Experience

UI Design / Wireframe

Description: Visual representation of your project's user interface and user experience design.

Deliverables:

Wireframes: Basic layout and structure

Mockups: Visual design with colors and typography

Prototypes: Interactive versions for user testing

Design System: Consistent components and patterns

Responsive Design: Mobile and desktop adaptations

Tools: 21st dev or V0

Resources & Tools

Development Tools

Version Control: Git, GitHub, GitLab

Remember: This template is a starting point. Adapt it to your specific project needs and hackathon requirements. The key is thorough preparation and clear communication of your project's value and potential.

Example Prompting

You are given a task to integrate an existing React component in the codebase

The codebase should support:

- shadcn project structure

- Tailwind CSS

- Typescript

If it doesn't, provide instructions on how to setup project via shadcn CLI, install Tailwind or Typescript.

Determine the default path for components and styles.

If default path for components is not /components/ui, provide instructions on why it's important to create this folder

Copy-paste this component to /components/ui folder:

```tsx

shape-landing-hero.tsx
```

```
"use client";

import { motion, useMotionValue, useTransform, animate } from "framer-motion";
import { Circle } from "lucide-react";
import { useEffect, useState } from "react";
import { cn } from "@/lib/utils";



function ElegantShape({
    className,
    delay = 0,
    width = 400,
    height = 100,
    rotate = 0,
    gradient = "from-white/[0.08]",
}: {
    className?: string;
    delay?: number;
    width?: number;
    height?: number;
    rotate?: number;
    gradient?: string;
}) {
    return (
        <motion.div
            initial={{
                opacity: 0,
                y: -150,
                rotate: rotate - 15,
```

```
    }}
    animate={{
      opacity: 1,
      y: 0,
      rotate: rotate,
    }}
    transition={{
      duration: 2.4,
      delay,
      ease: [0.23, 0.86, 0.39, 0.96],
      opacity: { duration: 1.2 },
    }}
    className={cn("absolute", className)}
  >
    <motion.div
      animate={{
        y: [0, 15, 0],
      }}
      transition={{
        duration: 12,
        repeat: Number.POSITIVE_INFINITY,
        ease: "easeInOut",
      }}
      style={{
        width,
        height,
      }}
      className="relative"
    >
```

```
            <div
              className={cn(
                "absolute inset-0 rounded-full",
                "bg-gradient-to-r to-transparent",
                gradient,
                "backdrop-blur-[2px] border-2 border-white/[0.15]",
                "shadow-[0_8px_32px_0_rgba(255,255,255,0.1)]",
                "after:absolute after:inset-0 after:rounded-full",
                "after:bg-[radial-gradient(circle_at_50%_50%,rgba(255,255,255,0.2),transparent_70%)]"
              )}
            />
          </motion.div>
        </motion.div>
    );
}

function HeroGeometric({
    badge = "Design Collective",
    title1 = "Elevate Your Digital Vision",
    title2 = "Crafting Exceptional Websites",
}: {
    badge?: string;
    title1?: string;
    title2?: string;
}) {
    const fadeUpVariants = {
        hidden: { opacity: 0, y: 30 },
        visible: (i: number) => ({
            opacity: 1,
```

```
        y: 0,
        transition: {
          duration: 1,
          delay: 0.5 + i * 0.2,
          ease: [0.25, 0.4, 0.25, 1],
        },
      }),
    };

    return (
      <div className="relative min-h-screen w-full flex items-center justify-center overflow-hidden bg-[#030303]">
        <div className="absolute inset-0 bg-gradient-to-br from-indigo-500/[0.05] via-transparent to-rose-500/[0.05] blur-3xl" />

        <div className="absolute inset-0 overflow-hidden">
          <ElegantShape
            delay={0.3}
            width={600}
            height={140}
            rotate={12}
            gradient="from-indigo-500/[0.15]"
            className="left-[-10%] md:left-[-5%] top-[15%] md:top-[20%]"
          />

          <ElegantShape
            delay={0.5}
            width={500}
            height={120}
```

```
    rotate={-15}
    gradient="from-rose-500/[0.15]"
    className="right-[-5%] md:right-[0%] top-[70%] md:top-[75%]"
/>

<ElegantShape
    delay={0.4}
    width={300}
    height={80}
    rotate={-8}
    gradient="from-violet-500/[0.15]"
    className="left-[5%] md:left-[10%] bottom-[5%] md:bottom-[10%]"
/>

<ElegantShape
    delay={0.6}
    width={200}
    height={60}
    rotate={20}
    gradient="from-amber-500/[0.15]"
    className="right-[15%] md:right-[20%] top-[10%] md:top-[15%]"
/>

<ElegantShape
    delay={0.7}
    width={150}
    height={40}
    rotate={-25}
    gradient="from-cyan-500/[0.15]"
```

```
            className="left-[20%] md:left-[25%] top-[5%] md:top-[10%]"
          />
        </div>

        <div className="relative z-10 container mx-auto px-4 md:px-6">
          <div className="max-w-3xl mx-auto text-center">
            <motion.div
              custom={0}
              variants={fadeUpVariants}
              initial="hidden"
              animate="visible"
              className="inline-flex items-center gap-2 px-3 py-1 rounded-full bg-white/[0.03] border
border-white/[0.08] mb-8 md:mb-12"
            >
              <Circle className="h-2 w-2 fill-rose-500/80" />
              <span className="text-sm text-white/60 tracking-wide">
                {badge}
              </span>
            </motion.div>

            <motion.div
              custom={1}
              variants={fadeUpVariants}
              initial="hidden"
              animate="visible"
            >
              <h1 className="text-4xl sm:text-6xl md:text-8xl font-bold mb-6 md:mb-8 tracking-tight">
                <span className="bg-clip-text text-transparent bg-gradient-to-b from-white to-
white/80">
```

```
                {title1}
              </span>
              <br />
              <span
                className={cn(
                  "bg-clip-text text-transparent bg-gradient-to-r from-indigo-300 via-white/90 to-rose-300 "
                )}
              >
                {title2}
              </span>
            </h1>
          </motion.div>

          <motion.div
            custom={2}
            variants={fadeUpVariants}
            initial="hidden"
            animate="visible"
          >
            <p className="text-base sm:text-lg md:text-xl text-white/40 mb-8 leading-relaxed font-light tracking-wide max-w-xl mx-auto px-4">
              Crafting exceptional digital experiences through
              innovative design and cutting-edge technology.
            </p>
          </motion.div>
        </div>
      </div>
```

```
        <div className="absolute inset-0 bg-gradient-to-t from-[#030303] via-transparent to-[#030303]/80 pointer-events-none" />
      </div>
    );
}

export { HeroGeometric }
```

demo.tsx

```
import { HeroGeometric } from "@/components/ui/shape-landing-hero"

function DemoHeroGeometric() {
  return <HeroGeometric badge="Kokonut UI"
        title1 = "Elevate Your"
        title2 = "Digital Vision" />
}

export { DemoHeroGeometric }
```

Install NPM dependencies:

```bash
lucide-react, framer-motion
```

Implementation Guidelines

1. Analyze the component structure and identify all required dependencies

2. Review the component's argumens and state

3. Identify any required context providers or hooks and install them

4. Questions to Ask

- What data/props will be passed to this component?

- Are there any specific state management requirements?

- Are there any required assets (images, icons, etc.)?

- What is the expected responsive behavior?

- What is the best place to use this component in the app?

Steps to integrate

0. Copy paste all the code above in the correct directories

1. Install external dependencies

2. Fill image assets with Unsplash stock images you know exist

3. Use lucide-react icons for svgs or logos if component requires them

Upwork Client

Summary

Squibler.io is hiring a no-code + full-stack builder who can work in Cursor and Lovable (or a comparable vibe-coding tool) and also write real code to deliver a production-ready Chatbot Generators Hub that supports programmatic SEO.

The experience will mirror a competitor’s public 'generators for writers' ecosystem while using Squibler branding and a clean, siloed information architecture. The person overseeing the project vibe-coded a MVP mockup for a competitor, and will be able to provide guidance on details.

The build will recreate and expand grouped “generator” families and their subpages:

Character names, book titles, plots, species/creatures, towns and cities, superpowers, pen names, and more.

Pages should feel focused and usable: a short prompt and simple selectors; generate/regenerate with a “something else” option; clear, copyable results and shareable URLs; concise explanatory content with examples/FAQs; and internal navigation that reinforces the silo (breadcrumbs, related links, hub links).

Quality expectations include indexable pages with a sensible URL structure, strong internal linking, fast and stable performance on mobile and desktop, accessible UI, meaningful analytics/event tracking, and robust error states. An admin/config approach should let non-technical teammates add or adjust generators without shipping code.

Initial focus: launch the Character Name Generators hub with a representative set of leaf pages across fantasy, historical, language, companions/pets, and archetypes. Follow-on phases will extend the same system and templates to species/creatures, towns and cities, superpowers, and pen names.

Required experience: power-user proficiency with Lovable (or similar) plus strong JavaScript/TypeScript and API skills; a track record of shipping functional apps (not just prototypes); and experience building SEO-friendly, scalable content systems.

Nice to have: familiarity with LLM integrations, caching and rate limiting, testing and accessibility, internationalization, and analytics instrumentation. Technical SEO and website information architecture knowledge is a bonus.

To apply, share links to shipped projects that demonstrate no-code and custom code working together; include a brief outline of how you would combine Lovable with custom code to achieve performance, analytics, and admin configurability at scale; and provide an estimate for the initial Character Name Generators milestone with your availability for follow-on work. Budget is flexible for the right builder.

Include the word "Pineapple" with your application. If you miss this word on the initial application, feel free to retry with the word pineapple included.

Scope of work:

1) Siloed SEO Architecture (to be recreated and expanded)

Top-level silos (each with a hub page and numerous subpages/variations):

Character Name Generators

Sub-silos to include all examples such as:

• Fantasy (e.g., fairy, sphinx, elemental, mutant-species)

• Historical/Ancient/Medieval (e.g., Anglo-Saxon, Gothic, Ancient Egyptian)

• Language & International (e.g., English, Swedish, Marathi, Japanese—Edo period)

• Companions & Pets (e.g., dogs, birds of prey, alien pets, puppets)

• Archetypes (hero, mentor, trickster, etc.)

Species / Creature Generators

Hub + subpages (e.g., Fantasy species, Alien species, Prehistoric creatures).

Town & City Name Generators

Hub + subpages for towns/cities and themed variants (e.g., city names, town names, Wild West, Fantasy towns, Roman towns, Elven cities, Goblin towns, Underwater cities, Cyberpunk cities).

Superpower Generator

Single hub with rich explainer content.

Pen Name Generators

Hub + genre pages (e.g., Fantasy, Science Fiction, Young Adult).

2) UX/Functionality (each generator page)

Inputs for short text prompts + optional selectors (genre/category), and # of results.

Generate / Regenerate actions; an alternate “something else” option for divergent results.

Result list/cards with copy-to-clipboard; optional save/favorite and shareable URLs (query-string state).

SEO content modules on every page: on-topic explainer, examples, etc.

Sibling navigation (related generators), breadcrumbs, and hub links to reinforce the silo.

Sign-up call to action that takes visitors to a Squibler signup page.

Required Experience:

Lovable and Cursor (or similar) no-code expertise plus strong coding skills (JavaScript/TypeScript, APIs; SSR/SSG in modern frameworks; serverless/edge patterns).

Proven track record shipping functional web apps (not just prototypes).

Experience building programmatic SEO systems (templates, data → pages, internal linking at scale).

UX implementation skills to match a provided design system/brand.