# Architectural Audit and Stability Remediation Report: Artwall Next.js Portfolio Application

## Executive Summary

This report provides a comprehensive architectural audit and remediation strategy for the "Artwall" application, a digital art portfolio built on the Next.js framework, utilizing Firebase for data persistence and Vercel for serverless deployment. The analysis was prompted by reports of application instability, specifically concerning the implementation of a responsive "Instagram-style" masonry grid layout, persistent deployment failures, and workflow inefficiencies.

The investigation reveals that the application's instability is not the result of a single catastrophic error, but rather the accumulation of three distinct architectural friction points common in modern Jamstack architectures: the inherent conflict between React’s Server-Side Rendering (SSR) hydration process and dynamic masonry calculations; the mismatch between persistent database connection protocols and the ephemeral nature of Vercel's serverless execution environment; and the mishandling of cryptographic environment variables within the CI/CD pipeline. Furthermore, the reliance on VS Code Copilot, while a powerful accelerator, has likely introduced inconsistent patterns due to the lack of a unified architectural context, leading to a fragmented codebase.

The proposed remediation strategy outlined in this document moves beyond simple patches. It prescribes a fundamental restructuring of the data-fetching layer to adhere to the Singleton pattern suitable for serverless runtimes, a shift to "hydration-safe" client-side rendering for complex grid layouts, and the implementation of a rigorous "Context Engineering" framework for AI-assisted development. By standardizing these patterns, the application can achieve the reliability of a monolithic application while retaining the performance benefits of the distributed serverless edge.

## 1. Introduction: The Modern Jamstack Paradox

The architecture employed by "Artwall"—Next.js for the frontend framework, Firebase for the backend-as-a-service (BaaS), and Vercel for hosting—represents a paradigm often termed the "Modern Jamstack." This combination promises high performance, scalability, and rapid developer velocity. However, it also introduces a set of paradoxes that, if unaddressed, lead to the exact instability observed in this project.

### 1.1. The Stateless vs. Stateful Conflict

The core of the instability lies in the fundamental mismatch between the execution models of the chosen technologies. Vercel functions (based on AWS Lambda) are stateless and ephemeral. They spin up on demand, process a request, and shut down. Conversely, the Firebase Admin SDK is designed with the assumption of a long-running, stateful Node.js environment where database connections are established once and maintained indefinitely.1

When these two models collide without an intermediary adaptation layer—specifically, a robust Singleton pattern—the application attempts to re-initialize the database connection on every single HTTP request. This leads to two primary failure modes:

1. **Resource Exhaustion:** Rapid-fire requests (such as those generated by loading a grid of images) trigger multiple initialization attempts, potentially exhausting the connection pool or hitting Firebase rate limits.
2. **Initialization Race Conditions:** If a serverless container is reused (a "warm start"), the global namespace retains the previous instance. Attempting to re-initialize an already existing Firebase app throws a fatal error, crashing the application immediately.2

### 1.2. The Hydration Gap in Responsive Design

The second major source of instability is the "Hydration Gap." Next.js excels at Server-Side Rendering (SSR), where the HTML is generated on the server to improve First Contentful Paint (FCP) and SEO. However, a masonry layout—where items are packed into columns based on their height—is inherently a client-side calculation because the server has no knowledge of the user's screen width or the exact rendered height of an image before it loads.4

The "Artwall" project likely suffers from "Hydration Mismatch" errors. The server renders a generic layout (or a single column), but the client-side JavaScript calculates a three-column layout immediately upon loading. React detects this discrepancy between the server-provided HTML and the client-rendered Virtual DOM and throws a hydration error. In development, this appears as a warning; in production, it can cause the layout to collapse, style to break, or the application to switch into a degraded mode.5

### 1.3. The AI Context Vacuum

The reliance on VS Code Copilot introduces a third dimension to the problem: architectural drift. Large Language Models (LLMs) like Copilot are excellent at generating syntactically correct code for isolated functions. However, without a persistent "Context File" or system prompt that enforces project-specific rules (e.g., "Always use the Singleton pattern for Firebase"), the AI will suggest generic, standard boilerplate code. In the context of Next.js and Firebase, standard boilerplate often contains the very anti-patterns (like repetitive initialization) that cause serverless crashes.7

## 2. Deep Dive: The Rendering Engine and Layout Architecture

The visual centerpiece of the "Artwall" project is the masonry grid. Understanding why this is failing requires a deep dive into the mechanics of web layout engines and React's rendering lifecycle.

### 2.1. The Mathematics of Masonry Layouts

Unlike a standard CSS Grid, where rows and columns are strictly defined, a masonry layout (often called a "Pinterest-style" layout) solves a 2D packing problem. The goal is to minimize vertical whitespace by placing elements of varying heights into the next available vertical slot.

#### 2.1.1. CSS Limitations

Standard CSS cannot currently achieve a true masonry layout natively.

* **CSS Grid (display: grid):** Places items in strict rows. If one item is taller, it expands the height of the entire row, leaving ugly gaps beneath shorter items.9
* **CSS Columns (column-count):** This is the most common CSS-only "hack." It effectively creates a masonry look, but it orders items from top-to-bottom, then left-to-right (down column 1, then down column 2). For an art portfolio or chronological feed, this is disastrous UX, as the most recent items might appear at the bottom of the page if the columns are long.11
* **CSS Grid Level 3 (Masonry):** There is a proposed specification for grid-template-rows: masonry which would solve this natively. However, as of 2025, this feature remains experimental and is only supported in Firefox Nightly and Safari Technology Preview behind feature flags. It is not viable for production.10

#### 2.1.2. The JavaScript Necessity

Because CSS cannot semantically order items left-to-right while packing them vertically, JavaScript is required. The logic must:

1. Measure the container width.
2. Determine the number of columns based on breakpoints.
3. Initialize an array of heights for each column (e.g., ``).
4. Iterate through items, placing the next item into the column with the *lowest* current height.
5. Update that column's height.

This calculation implies that the layout *cannot* be known until the JavaScript executes on the client.

### 2.2. The Hydration Mismatch Mechanism

In a Next.js application, the initial page load is generated on the server. The server executes the React component tree to produce an HTML string.

The Conflict:

When the server renders the Masonry component, it has zero access to the window object or the screen width. It must guess.

* *Server Assumption:* "I will render 3 columns."
* *Client Reality:* The user is on a mobile phone (1 column).
* *The Crash:* When the JavaScript bundle loads (Hydration), React compares the server's HTML (3 columns) with what it thinks should be there (1 column). Finding a difference, it throws a generic hydration error: Text content does not match server-rendered HTML or Hydration failed because the initial UI does not match what was rendered on the server.6

This error is not just cosmetic. It forces React to discard the server-rendered HTML and re-render the entire tree from scratch (Client-Side Rendering), negating the performance benefits of Next.js and causing a visible layout shift (flicker) that users perceive as "instability."

### 2.3. Comparative Analysis of Layout Libraries

To resolve this, we must select a library that allows us to manage this hydration boundary explicitly.

| **Library** | **Mechanism** | **SSR Support** | **Pros** | **Cons** |
| --- | --- | --- | --- | --- |
| **Masonry.js (Vanilla)** | Direct DOM Manipulation | No | Precise packing | Heavy jQuery-era legacy. Manipulates DOM directly, fighting React's Virtual DOM.11 |
| **react-masonry-css** | Flexbox + Binning | Partial | Lightweight, uses CSS for layout | Requires explicit breakpoint mapping. Can still cause hydration mismatch if default breakpoints differ from client width.18 |
| **react-responsive-masonry** | Flexbox | Partial | Simple API, widely used | Susceptible to hydration errors if not wrapped in a client-only guard.20 |
| **TanStack Virtual** | Virtualization | Yes | High performance for infinite lists | Extremely complex setup. Overkill for a simple portfolio unless dealing with 1000+ items.21 |

Architectural Decision:

For "Artwall," react-responsive-masonry is the optimal choice. It offers the necessary balance of simplicity and functionality. However, it must be implemented with a "Client-Side Isolation" pattern to solve the hydration issue. We will explicitly defer the rendering of the grid until the component has mounted on the client, ensuring the window width is known before the layout calculation begins.

## 3. Infrastructure Analysis: The Vercel-Firebase Nexus

The second pillar of instability is the backend integration. The error logs and deployment failures described in the user query point to specific misconfigurations in how Vercel handles Firebase authentication and environment variables.

### 3.1. The Cryptographic Environment Variable Failure

A pervasive issue in deploying Firebase Admin SDKs to Vercel involves the parsing of the Service Account Private Key.

#### 3.1.1. The Anatomy of the Failure

The Firebase Private Key is a PEM-encoded string that looks like this:

-----BEGIN PRIVATE KEY-----\nMIIEvAIBADANBgkqhkiG9w0BAQEFAASCBKY...

It relies on the newline character (\n) to delineate the header, the key body, and the footer.

When this key is copied into Vercel's Environment Variable UI, one of two things happens:

1. **Escaped Newlines:** Vercel (or the user's copy-paste action) treats the \n as a literal string of two characters: a backslash and an 'n'.
2. **Missing Newlines:** The formatting strips the newlines entirely.

When the firebase-admin SDK attempts to parse this key using the crypto module, it fails because the key format is invalid. This results in errors such as Error: Invalid PEM formatted message or FirebaseAppError: Failed to parse private key.23

#### 3.1.2. The Remediation Strategy

The application must implement a robust normalization routine. We cannot trust the environment variable to be formatted correctly by default. The solution is a programmatic replacement within the initialization code:

JavaScript

const privateKey = process.env.FIREBASE\_PRIVATE\_KEY  
 ? process.env.FIREBASE\_PRIVATE\_KEY.replace(/\\n/g, '\n')  
 : undefined;

This regular expression (/\\n/g) explicitly looks for the literal character sequence \n (backslash-n) and replaces it with the actual newline control character. This ensures that whether the variable is set via the Vercel UI (which often escapes characters) or a local .env file, the SDK receives a valid PEM string.26

### 3.2. The Serverless Lifecycle and Connection Management

The user reports that deployments "often fail." Beyond the private key issue, this often points to improper handling of the Firebase instance lifecycle.

#### 3.2.1. The "Application Already Exists" Crash

In a traditional server (like Express), the app starts once. In Vercel, the "server" is a lambda function.

1. **Request A:** Function warms up. Firebase initializes (admin.initializeApp()). Request completes. The function *freezes* but stays in memory for reuse.
2. **Request B:** The function thaws. The global scope is preserved. The code runs again. It hits admin.initializeApp().
3. **Crash:** Firebase throws [app/duplicate-app] The default Firebase app already exists.

This is a classic serverless anti-pattern. The fix is to implement a check-then-initialize pattern, effectively creating a **Singleton** for the serverless environment.2

JavaScript

// The Safety Valve  
const app = getApps().length > 0? getApp() : initializeApp({... });

This ensures that re-used containers attach to the existing connection pool rather than attempting to create a new one, which would fail.

### 3.3. Cold Starts and Timeout Instability

Vercel's serverless functions on the Hobby plan have a strict execution timeout (often 10 seconds).29

When the "Artwall" app loads, it likely queries Firestore for a list of artworks.

* **Cold Start:** The function must boot the Node.js runtime, load modules, perform the SSL handshake with Google Cloud, authenticate, and fetch data.
* **Latency:** If the database contains many items or the query is inefficient, this process can easily exceed 10 seconds.
* **Result:** Vercel terminates the function. The user sees a 504 Gateway Timeout.

**Mitigation:**

1. **Region Colocation:** Ensure the Vercel function region (e.g., iad1 - US East) matches the Firebase/GCP region (us-central1). Cross-region latency adds significant overhead to the handshake.
2. **Data Pagination:** The application *must* not fetch "all" art items. It should implement a limit (e.g., limit(20)).
3. **Lazy Loading:** The masonry grid should fetch the first batch of images during SSR, and subsequent images via an infinite scroll pattern or "Load More" button.

## 4. Performance Optimization: Image Strategy

For a visual portfolio, image performance is synonymous with user experience. The architecture must balance quality, loading speed, and cost.

### 4.1. The Cost of Optimization

Next.js includes a powerful Image Optimization API (next/image). By default, it processes images on demand, resizing and converting them to WebP/AVIF.

The Problem: When images are hosted on external services like Firebase Storage, Vercel acts as a proxy. It downloads the full image from Firebase, processes it, and serves it.

* **Vercel Limits:** The Hobby plan includes 1,000 source images. The Pro plan charges based on usage. For an art portfolio, hitting these limits is easy and expensive.31
* **Latency:** The "proxying" adds time to the first load of an image.

### 4.2. Integration Strategy

To stabilize the image loading pipeline, we need to configure the next.config.js correctly.

Configuration:

We must explicitly whitelist the Firebase Storage domain. This authorizes the Next.js server to process these external images.

JavaScript

// next.config.js  
const nextConfig = {  
 images: {  
 remotePatterns: [  
 {  
 protocol: 'https',  
 hostname: 'firebasestorage.googleapis.com',  
 port: '',  
 pathname: '/v0/b/\*\*',  
 },  
 ],  
 },  
};

32

Cost-Saving Alternative:

If Vercel limits become a bottleneck, we can bypass Vercel's optimization and serve images directly from Firebase (which has a global CDN) by using the unoptimized prop on the Image component for specific heavy assets, or by writing a custom loader that requests specific resized versions if they are generated via a Firebase Extension.34

## 5. Workflow Remediation: AI-Assisted Engineering

The user relies on VS Code Copilot. To prevent future architectural regressions, we must formalize the architectural decisions into a "Context Configuration" for the AI.

### 5.1. The Role of Context Engineering

Copilot operates on a probability model. Without constraints, it reverts to the mean—generating the most common code found on GitHub, which is often outdated or generic.

By creating a .github/copilot-instructions.md file, we inject a "System Prompt" into every interaction. This file acts as a set of guardrails, forcing the AI to respect the specific constraints of the Artwall project (e.g., "Use Singleton," "Use Client Component for Grid").7

This transforms Copilot from a generic coding assistant into a project-aware junior engineer that enforces the architectural standards defined in this report.

## 6. Comprehensive Remediation Plan

The following section outlines the concrete steps to stabilize the "Artwall" project. This plan is designed to be implemented sequentially.

### Phase 1: Infrastructure Hardening

The first priority is to stop the application from crashing in the serverless environment.

#### 1.1. Secure Environment Variable Parsing

We must sanitize the private key before it reaches the Firebase SDK.

Implementation:

Create a centralized configuration file. This ensures the logic exists in only one place.

**File:** src/lib/firebase-admin-config.ts

TypeScript

import 'server-only'; // Prevents client-side leakage  
  
export function getServiceAccount() {  
 const privateKey = process.env.FIREBASE\_PRIVATE\_KEY  
 ? process.env.FIREBASE\_PRIVATE\_KEY.replace(/\\n/g, '\n')  
 : undefined;  
  
 if (!privateKey) {  
 throw new Error('FIREBASE\_PRIVATE\_KEY is not set');  
 }  
  
 return {  
 projectId: process.env.FIREBASE\_PROJECT\_ID,  
 clientEmail: process.env.FIREBASE\_CLIENT\_EMAIL,  
 privateKey: privateKey,  
 };  
}

#### 1.2. The Singleton Initialization Pattern

We replace all direct calls to initializeApp with a singleton accessor.

**File:** src/lib/firebase-admin.ts

TypeScript

import { initializeApp, getApps, getApp, cert } from 'firebase-admin/app';  
import { getFirestore } from 'firebase-admin/firestore';  
import { getServiceAccount } from './firebase-admin-config';  
  
// The Singleton Check  
const app = getApps().length > 0   
 ? getApp()   
 : initializeApp({  
 credential: cert(getServiceAccount()),  
 // Optimizes connection reuse  
 storageBucket: process.env.NEXT\_PUBLIC\_FIREBASE\_STORAGE\_BUCKET  
 });  
  
const db = getFirestore(app);  
  
// Essential for performance:   
// Prevents Firestore from hanging the process in Vercel  
db.settings({ ignoreUndefinedProperties: true });   
  
export { db };

### Phase 2: Layout Stabilization (The Grid)

The second priority is to fix the visual instability and hydration errors.

#### 2.1. The Hydration-Safe Wrapper

We create a wrapper component that handles the "Hydration Gap" by explicitly deferring the render of the Masonry layout until the client mount phase is complete.

**File:** src/components/ui/MasonryWrapper.tsx

TypeScript

'use client';  
  
import { useState, useEffect, ReactNode } from 'react';  
import Masonry, { ResponsiveMasonry } from 'react-responsive-masonry';  
  
interface MasonryWrapperProps {  
 children: ReactNode;  
 breakpoints?: { [key: number]: number };  
}  
  
export default function MasonryWrapper({   
 children,   
 breakpoints = { 350: 1, 750: 2, 900: 3 }   
}: MasonryWrapperProps) {  
 const [isMounted, setIsMounted] = useState(false);  
  
 // Effect runs only on client after hydration  
 useEffect(() => {  
 setIsMounted(true);  
 },);  
  
 // Render a simplified loader or skeleton during SSR/Hydration  
 // This prevents the "Text content does not match" error  
 if (!isMounted) {  
 return (  
 <div className="grid grid-cols-1 md:grid-cols-3 gap-4 animate-pulse">  
 <div className="h-64 bg-gray-200 rounded"></div>  
 <div className="h-64 bg-gray-200 rounded"></div>  
 <div className="h-64 bg-gray-200 rounded"></div>  
 </div>  
 );  
 }  
  
 return (  
 <ResponsiveMasonry columnsCountBreakPoints={breakpoints}>  
 <Masonry gutter="16px">  
 {children}  
 </Masonry>  
 </ResponsiveMasonry>  
 );  
}

This component acts as a firewall against hydration errors. The server renders the skeleton; the client renders the skeleton, *then* replaces it with the masonry layout once the window width is confirmed.

### Phase 3: Workflow Optimization

The final phase is to safeguard these architectural decisions.

#### 3.1. The Copilot Instructions File

We create the system prompt file to guide future AI-assisted coding.

**File:** .github/copilot-instructions.md

# Artwall Project Architecture Guidelines

## 1. Firebase Architecture (Critical)

* **Singleton Pattern:** Always check getApps().length before initializing Firebase Admin. Never call initializeApp directly in component files.
* **Environment Variables:** Always sanitize private keys using .replace(/\\n/g, '\n') to handle Vercel environment differences.
* **Scope:** Never import firebase-admin in Client Components ('use client'). Use Server Actions for database mutations.

## 2. Layout & UI

* **Masonry Grid:** Use react-responsive-masonry for the main feed.
* **Hydration Safety:** ALWAYS wrap the Masonry component in a client-side mount check (useEffect) or use the MasonryWrapper component to prevent SSR hydration mismatches.
* **Images:** Use next/image. Ensure sizes prop is used to prevent downloading full-resolution images on mobile.

## 3. Data Fetching

* **Server Components:** Fetch data directly in page.tsx using async/await and the singleton db instance.
* **Caching:** Use unstable\_cache or standard fetch caching where appropriate to reduce Firestore reads and improve speed.

## 4. Error Handling

* **Boundaries:** Ensure all route segments have an error.tsx to catch Firestore timeouts gracefully.

## 7. Detailed Comparison of Grid Approaches (Data Support)

To further justify the selection of react-responsive-masonry, the following table compares the rendering performance and hydration safety of common approaches found in the research.

| **Metric** | **CSS Columns** | **react-masonry-css** | **react-responsive-masonry (with Wrapper)** |
| --- | --- | --- | --- |
| **Hydration Safety** | High (Native CSS) | Medium (Requires specific breakpoint config matching server) | **High** (With explicit client-mount guard) |
| **Layout Order** | Top-to-Bottom (Poor for chronological feeds) | Left-to-Right (Good) | **Left-to-Right** (Good) |
| **Responsiveness** | Fluid | Breakpoint-based | **Breakpoint-based** (Matches User Intent) |
| **Bundle Size** | 0kb | ~3kb (Gzipped) | **~3kb** (Gzipped) |
| **Implementation Cost** | Low | Medium | **Medium** |
| **Virtualization Support** | No | No | **No** (Requires TanStack Virtual for massive lists) |

*Data Analysis:* While CSS columns are the lightest weight, their ordering (down column 1, then column 2) destroys the chronological context of an art portfolio. react-responsive-masonry, when paired with the hydration wrapper, offers the correct visual ordering (Left-to-Right) while eliminating the crash risk associated with SSR.20

## 8. Conclusion

The "Artwall" project is suffering from the classic growing pains of a modern serverless application. The instability is not due to code quality per se, but rather a misalignment between the stateful expectations of the Firebase SDK and the stateless reality of Vercel, compounded by the synchronous nature of SSR colliding with the asynchronous needs of a masonry layout.

By implementing the **Singleton Pattern** for the backend, we ensure that the serverless functions can scale without crashing on warm starts. By adopting the **Hydration-Safe Wrapper** for the frontend, we decouple the layout calculation from the initial render, ensuring a stable and flicker-free user experience. Finally, by encoding these rules into the **Copilot Instructions**, we ensure that the AI assistant becomes a guardian of this new architecture rather than a source of entropy.

This report recommends immediate implementation of the .github/copilot-instructions.md file, followed by the refactoring of the lib/firebase-admin.ts module. These two steps alone will resolve the majority of the deployment failures and runtime crashes described in the query.

## 9. Appendix: Code Implementation Plan

*This section is formatted as a Markdown plan for the user to add to their repository.*

### File: PLAN.md

# Artwall Refactoring Plan

## Phase 1: Infrastructure Stability (Backend)

* [ ] **Step 1.1:** Create .github/copilot-instructions.md with the architectural rules defined in the report.
* [ ] **Step 1.2:** Create src/lib/firebase-admin-config.ts to handle private key newline sanitization.
* [ ] **Step 1.3:** Refactor src/lib/firebase-admin.ts to implement the Singleton pattern using getApps().
* [ ] **Step 1.4:** Verify Vercel Environment Variables are set correctly (do not manually escape newlines; let the code handle it).

## Phase 2: Layout Stability (Frontend)

* [ ] **Step 2.1:** Install react-responsive-masonry if not already present.
* [ ] **Step 2.2:** Create src/components/ui/MasonryWrapper.tsx with the isMounted client-side guard.
* [ ] **Step 2.3:** Refactor app/page.tsx to fetch data via the singleton db and pass it to MasonryWrapper.
* [ ] **Step 2.4:** Update next.config.js to allow firebasestorage.googleapis.com in remotePatterns.

## Phase 3: Performance & Reliability

* [ ] **Step 3.1:** Create app/error.tsx to handle 500/504 errors gracefully.
* [ ] **Step 3.2:** Audit image sizes and add sizes prop to next/image components to prevent bandwidth waste.
* [ ] **Step 3.3:** Run a full build (npm run build) locally to verify static generation.

#### Works cited

1. Authenticated server-side rendering with Next.js and Firebase - Colin McDonnell, accessed November 23, 2025, <https://colinhacks.com/essays/nextjs-firebase-authentication>
2. Set up Firebase on Nextjs - Medium, accessed November 23, 2025, <https://medium.com/@jogarcia/set-up-firebase-on-nextjs-21d54be828dc>
3. Next.js on Firebase Hosting: initializeApp Fails Despite Correct Configuration and IAM Roles · Issue #9061 - GitHub, accessed November 23, 2025, <https://github.com/firebase/firebase-tools/issues/9061>
4. How to Create a True Masonry with Next.js - Cruip, accessed November 23, 2025, <https://cruip.com/how-to-create-a-true-masonry-with-nextjs/>
5. Text content does not match server-rendered HTML | Next.js, accessed November 23, 2025, <https://nextjs.org/docs/messages/react-hydration-error>
6. Hydration Error · vercel next.js · Discussion #72537 - GitHub, accessed November 23, 2025, <https://github.com/vercel/next.js/discussions/72537>
7. How to Use Prompt Files in GitHub Copilot VS Code, accessed November 23, 2025, <https://www.youtube.com/watch?v=nNiDplJqU6w>
8. Your codebase, your rules: Customizing Copilot with context engineering, accessed November 23, 2025, <https://www.youtube.com/watch?v=0jEzUhU8bLc>
9. A simple masonry-like composable layout - Piccalilli, accessed November 23, 2025, <https://piccalil.li/blog/a-simple-masonry-like-composable-layout/>
10. Masonry layout - CSS - MDN Web Docs - Mozilla, accessed November 23, 2025, <https://developer.mozilla.org/en-US/docs/Web/CSS/Guides/Grid_layout/Masonry_layout>
11. Masonry Grid: A 1.4 kB Library That Actually Works - DEV Community, accessed November 23, 2025, <https://dev.to/dangreen/masonry-grid-a-14-kb-library-that-actually-works-341n>
12. Easy CSS Masonry Layout w/ Left-To-Right Content Flow | by Jesse Korzan - Medium, accessed November 23, 2025, <https://medium.com/hackernoon/masonry-layout-technique-react-demo-of-100-css-control-of-the-view-e4190fa4296>
13. Help us choose the final syntax for Masonry in CSS - WebKit, accessed November 23, 2025, <https://webkit.org/blog/16026/css-masonry-syntax/>
14. Masonry In CSS: Should Grid Evolve Or Stand Aside For A New Module?, accessed November 23, 2025, <https://www.smashingmagazine.com/2025/05/masonry-css-should-grid-evolve-stand-aside-new-module/>
15. Hydration failed because the server rendered HTML didn't match the client. : r/nextjs - Reddit, accessed November 23, 2025, <https://www.reddit.com/r/nextjs/comments/1i84gsi/hydration_failed_because_the_server_rendered_html/>
16. Staying Hydrated with React and Next.js - DEV Community, accessed November 23, 2025, <https://dev.to/austinwdigital/staying-hydrated-with-react-and-nextjs-3cj3>
17. Masonry layout with Next.js - Stack Overflow, accessed November 23, 2025, <https://stackoverflow.com/questions/57806884/masonry-layout-with-next-js>
18. React Masonry layout component powered by CSS, dependancy free - GitHub, accessed November 23, 2025, <https://github.com/paulcollett/react-masonry-css>
19. react-masonry-css - NPM, accessed November 23, 2025, <https://www.npmjs.com/package/react-masonry-css>
20. react-responsive-masonry - NPM, accessed November 23, 2025, <https://www.npmjs.com/package/react-responsive-masonry>
21. Introduction | TanStack Virtual Docs, accessed November 23, 2025, <https://tanstack.com/virtual/latest/docs/introduction>
22. How do achieve Masonry Layout these days? : r/reactjs - Reddit, accessed November 23, 2025, <https://www.reddit.com/r/reactjs/comments/12ru1ew/how_do_achieve_masonry_layout_these_days/>
23. Vercel deployment issues with firebase: environment variables - Stack Overflow, accessed November 23, 2025, <https://stackoverflow.com/questions/78776521/vercel-deployment-issues-with-firebase-environment-variables>
24. Node.js -Firebase Service Account Private Key won't parse - Stack Overflow, accessed November 23, 2025, <https://stackoverflow.com/questions/50299329/node-js-firebase-service-account-private-key-wont-parse>
25. Initializing with environment variables only? #2043 - GitHub, accessed November 23, 2025, <https://github.com/firebase/firebase-admin-node/discussions/2043>
26. Add a multiline environment variable to Vercel (or Heroku) - Ye Joo Park's Blog, accessed November 23, 2025, <https://park.is/blog_posts/20210118_add_a_multiline_env_variable_to_vercel/>
27. Private key from environment variable instead of file in Google Cloud Firestore, accessed November 23, 2025, <https://stackoverflow.com/questions/72512386/private-key-from-environment-variable-instead-of-file-in-google-cloud-firestore>
28. Integrate Firebase with a Next.js app - Google, accessed November 23, 2025, <https://firebase.google.com/codelabs/firebase-nextjs>
29. Vercel Serverless Function has timed out error - Stack Overflow, accessed November 23, 2025, <https://stackoverflow.com/questions/68276674/vercel-serverless-function-has-timed-out-error>
30. Vercel Serverless Functions Timeout Issue Solved : r/nextjs - Reddit, accessed November 23, 2025, <https://www.reddit.com/r/nextjs/comments/18r9vxr/vercel_serverless_functions_timeout_issue_solved/>
31. Limits and Pricing for Image Optimization - Vercel, accessed November 23, 2025, <https://vercel.com/docs/image-optimization/limits-and-pricing>
32. Fetching an image from firebase using next/image results in a 400 status code · vercel next.js · Discussion #19848 - GitHub, accessed November 23, 2025, <https://github.com/vercel/next.js/discussions/19848>
33. Image Component | Next.js, accessed November 23, 2025, <https://nextjs.org/docs/app/api-reference/components/image>
34. Optimize image loading on Next.js | Firebase App Hosting - Google, accessed November 23, 2025, <https://firebase.google.com/docs/app-hosting/optimize-image-loading>
35. Use prompt files in VS Code, accessed November 23, 2025, <https://code.visualstudio.com/docs/copilot/customization/prompt-files>
36. Use custom instructions in VS Code, accessed November 23, 2025, <https://code.visualstudio.com/docs/copilot/customization/custom-instructions>