# Website Performance Optimization Report

**To:** Frontend Developer

**From:** Manus AI

**Date:** December 11, 2025

**Subject:** Comprehensive Performance Optimization Recommendations for Amtalek Website

## 1. Executive Summary

This report provides a detailed analysis of the Amtalek website's frontend performance, identifies key bottlenecks, and offers actionable recommendations for improvement. The primary issues identified are slow API response times, unoptimized images, and a large number of resource requests. By implementing the recommendations in this report, you can significantly improve page load times, enhance the user experience, and boost the overall performance of the website.

## 2. Performance Analysis

### 2.1. Overall Performance Metrics

A performance analysis of the website was conducted, and the following key metrics were collected:

| Metric | Value |
|---|---|
| Time to First Byte (TTFB) | 1604 ms |
| DOM Interactive | 3627 ms |
| DOM Complete | 4000 ms |
| Load Complete | 4000 ms |
| Total Resources | 95 |
| Total DOM Nodes | 1108 |
| Images on Page | 23 |
| Scripts on Page | 50 |

These metrics indicate that the website takes a significant amount of time to load, with a high number of resources and DOM nodes contributing to the slow performance.

### 2.2. Key Performance Issues

The following key performance issues were identified:

- **Slow API Responses:** Several API calls to `newdashboard.amtalek.com` are taking a long time to complete, with some requests exceeding 2.5 seconds. This significantly delays the rendering of content on the page.
- **Unoptimized Images:** Many images on the website are not optimized for the web. They are served in formats like PNG and JPEG instead of modern formats like WebP, and they are not properly sized for their containers. This results in larger file sizes and slower load times.
- **Excessive Resource Requests:** The website makes a large number of requests for CSS, JavaScript, and other resources. This increases the overall load time and can block the rendering of the page.
- **Large DOM Size:** The page has a large number of DOM nodes, which can slow down rendering and increase memory usage.

## 3. Actionable Recommendations

To address the identified performance issues, the following recommendations are provided:

### 3.1. Image Optimization

- **Use Next-Gen Image Formats:** Convert images to modern formats like WebP, which offer better compression and smaller file sizes. The Next.js `<Image>` component can automatically optimize images to WebP format `1`.
- **Properly Size Images:** Ensure that images are sized appropriately for their containers. The Next.js `<Image>` component can help with this by providing `width` and `height` props.
- **Implement Lazy Loading:** Use lazy loading for images that are not in the initial viewport. This will defer the loading of off-screen images until they are needed, improving the initial page load time. The Next.js `<Image>` component has built-in lazy loading capabilities `2`.

### 3.2. Resource Optimization

- **Bundle and Minify Resources:** Bundle and minify your CSS and JavaScript files to reduce the number of requests and the overall file size. Next.js automatically handles

this for you, but it's important to ensure that you are not inadvertently disabling these optimizations.

- **Defer Non-Critical Scripts:** Defer the loading of non-critical JavaScript files to prevent them from blocking the rendering of the page. The Next.js `<Script>` component provides a `strategy` prop that can be used to control when a script is loaded [3].

### 3.3. API Performance

- **Optimize Backend API Calls:** Work with the backend team to optimize the performance of the API calls. This may involve optimizing database queries, caching responses, or using a more efficient API architecture.
- **Reduce the Number of API Requests:** Where possible, reduce the number of API requests by combining multiple requests into a single request.

### 3.4. Rendering Performance

- **Implement Code Splitting:** Use dynamic imports to split your code into smaller chunks. This will allow the browser to load only the code that is needed for the current page, improving the initial load time.
- **Use Server-Side Rendering (SSR):** Use server-side rendering to pre-render pages on the server. This will improve the perceived performance of the website by showing the user content sooner.

## 4. Prioritized Action Plan

The following table prioritizes the recommendations based on their impact and effort:

| Priority | Recommendation | Impact | Effort |
|---|---|---|---|
| High | Optimize Backend API Calls | High | High |
| High | Use Next-Gen Image Formats | High | Medium |
| High | Properly Size Images | High | Medium |
| Medium | Implement Lazy Loading | Medium | Low |
| Medium | Defer Non-Critical Scripts | Medium | Low |

| | | | |
|---|---|---|---|
| Medium | Reduce the Number of API Requests | Medium | Medium |
| Low | Implement Code Splitting | Low | Medium |
| Low | Use Server-Side Rendering (SSR) | Low | High |

## 5. Conclusion

By implementing the recommendations in this report, you can significantly improve the performance of the Amtalek website. This will lead to a better user experience, increased engagement, and improved search engine rankings. It is recommended to start with the high-priority items to achieve the most significant performance gains.

## 6. References

[1] Next.js Documentation - Image Optimization:
[2] Next.js Documentation - Image Component:
[3] Next.js Documentation - Script Component: