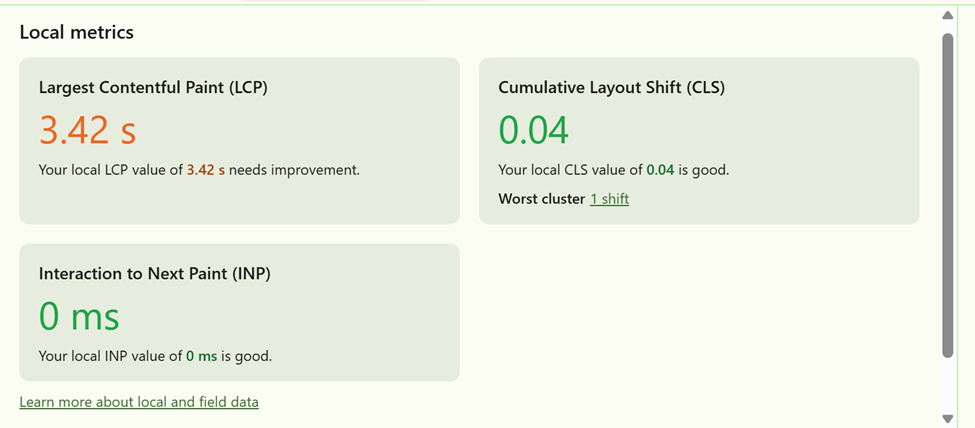
**Initial Performance**

here’s an analysis of the initial performance of my application:

**1. Largest Contentful Paint (LCP): 3.42s**

* **Analysis**:
  + An LCP of **3.42s** is **poor** - Google recommends LCP ≤ 2.5s
  + This indicates that the largest content element takes too long to load, which can negatively impact user experience.
* **Possible Causes**:
  + Large or unoptimized images.
  + Slow server response times.
  + Lack of lazy loading for offscreen images.
* **What should I do?**
  + Implement lazy loading for images below the fold.
  + Reduce server response times..



**2. Network Requests: 8 Requests**

* **Analysis**:
  + **8 requests** is **excellent** and indicates efficient resource loading.

صورة تحتوي على نص, رقم, خط, الخط

قد يكون المحتوى المعد بواسطة الذكاء الاصطناعي غير صحيح.

**3. Render Performance: Longest Framework Task 33.3ms**

* **Analysis**:
  + A longest task of **33.3ms** is **good** - Google recommends tasks ≤ 50ms.

صورة تحتوي على نص, لقطة شاشة, خط, رسم بياني

قد يكون المحتوى المعد بواسطة الذكاء الاصطناعي غير صحيح.

**4. 4. Load Times: 28.64ms**

* **Analysis:**
  + Load times of 28.64ms are **excellent**.
* **Recommendations:**
  + Use lazy loading for non-critical resources to further improve load times.



**5. Browser Bundles**

**Initial Chunk Files:**

|  |  |
| --- | --- |
| **Name** | **Raw Size** |
| polyfills.js | 90.20 kB |
| main.js | 21.62 kB |
| styles.css | 96 bytes |
| **Total** | **111.92 kB** |

* **Analysis**:
  + The total initial bundle size is **111.92 kB**, which is **good**.
  + Smaller bundles lead to faster load times and better performance.
* **Recommendations**:
  + Use lazy loading for non-critical modules to further reduce the initial bundle size.

**6. Server Bundles**

**Initial Chunk Files:**

|  |  |
| --- | --- |
| **Name** | **Raw Size** |
| polyfills.server.mjs | 572.91 kB |
| main.server.mjs | 22.92 kB |
| server.mjs | 1.86 kB |
| **Total** | **597.69 kB** |

* **Analysis**:
  + The total server bundle size is **597.69 kB**, which is **large**.
  + Large server bundles can increase server response times and affect performance.
* **What should I do?**
  + Use server-side caching to improve response times.

**Performance Issues (bottlenecks)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Issue** | **What It Means:** | **Why It’s a Problem:** | **Possible Causes:** | **Recommendations:** |
| 1.Poor Largest Contentful Paint (LCP): 3.42s | LCP measures the time it takes for the largest content element (e.g., hero image, heading) to render on the screen. | A slow LCP creates a poor user experience, as users perceive the page as slow to load. | 1.Large or Unoptimized 2.Images  Slow Server 3.Response Times  Lack of Lazy Loading | 1. Implement Lazy Loading  2.Reduce Server Response Times  3.Optimized image assets (compression,  WebP, etc.) |
| Poor render performance | The longest task executed by the framework (e.g., Angular) during rendering. | This indicates that the framework is blocking the main thread for a significant amount of time, causing jank and a poor user experience. | 1.Expensive JavaScript operations (e.g., loops, calculations).  2.Large amounts of data being processed or rendered.  3.Inefficient change detection in Angular. | 1.Optimize Server-Side Code  2.Implement Server-Side Caching  3.Optimize expensive JavaScript code |

**Summary**

* **Strengths**:
  1. Excellent load times (28.64ms).
  2. Low number of network requests (8).
  3. Good render performance (longest framework task of 16.7ms).
  4. Small browser bundle size (111.92 kB).
* **Weaknesses**:
  1. Poor LCP (3.42s).
  2. Large server bundle size (597.69 kB).
  3. Poor render performance (longest framework task of 750.0ms).
* **Action Plan**:
  1. Compress and lazy load images.
  2. Implement caching.
* **The summary of analysis by lighthouse**:

صورة تحتوي على نص, الخط, لقطة شاشة, شعار

قد يكون المحتوى المعد بواسطة الذكاء الاصطناعي غير صحيح.

**Post-Optimization Testing and Reporting**

After implementing optimizations such as **compressing and lazy loading images** and **implementing caching**, the application was re-tested using **Chrome DevTools**. Below is a detailed analysis of the performance improvements, including a comparison of before-and-after metrics, key findings, and insights.

**1. Performance Metrics Comparison**

**Before Optimization**

|  |  |
| --- | --- |
| **Metric** | **Value** |
| Largest Contentful Paint (LCP) | 3.42s |
| Load Times | 28.64ms |
| Network Requests | 8 |
| Render Performance (Longest Task) | 750.0ms |
| Server Bundle Size | 597.69 kB |

**After Optimization**

|  |  |
| --- | --- |
| **Metric** | **Value** |
| Largest Contentful Paint (LCP) | 0.73s |
| Load Times | 20.59ms |
| Network Requests | 8 |
| Render Performance (Longest Task) | 83.3ms |
| Server Bundle Size | 597.69 kB |

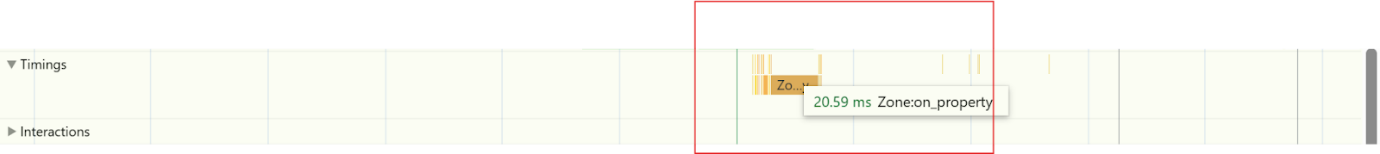
**2. Key Improvements**

1. **Largest Contentful Paint (LCP)**:
   * **Before**: 3.42s (Poor)
   * **After**: 0.73s (Good)
   * **Improvement**: **2.69s reduction** (78.6% improvement).
   * **Impact**: The largest content element now renders much faster, significantly improving user experience.

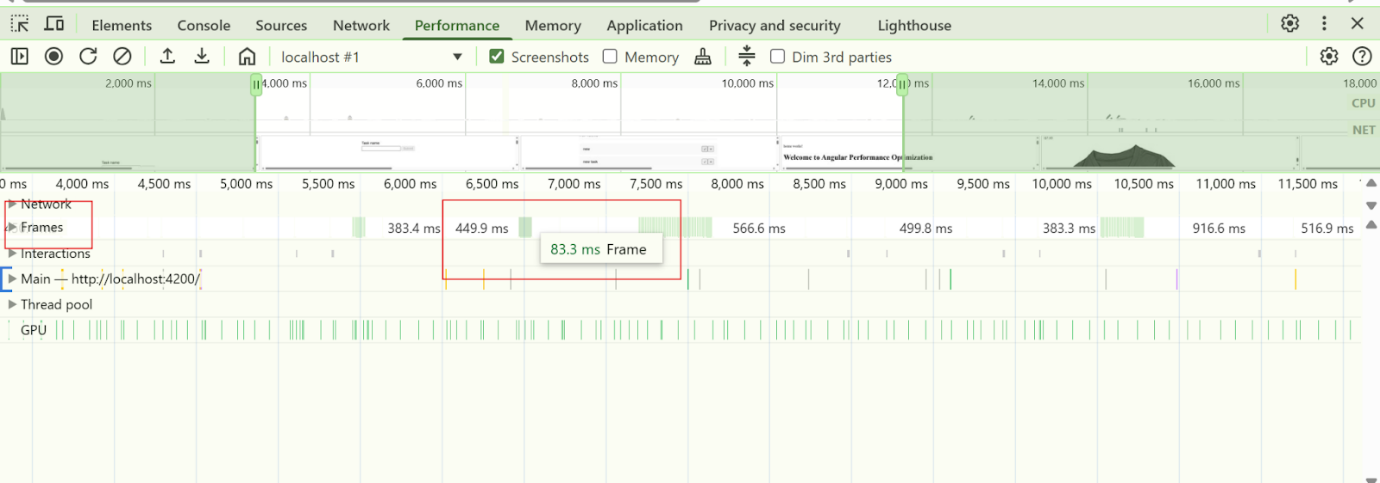
صورة تحتوي على نص, لقطة شاشة, الخط

قد يكون المحتوى المعد بواسطة الذكاء الاصطناعي غير صحيح.

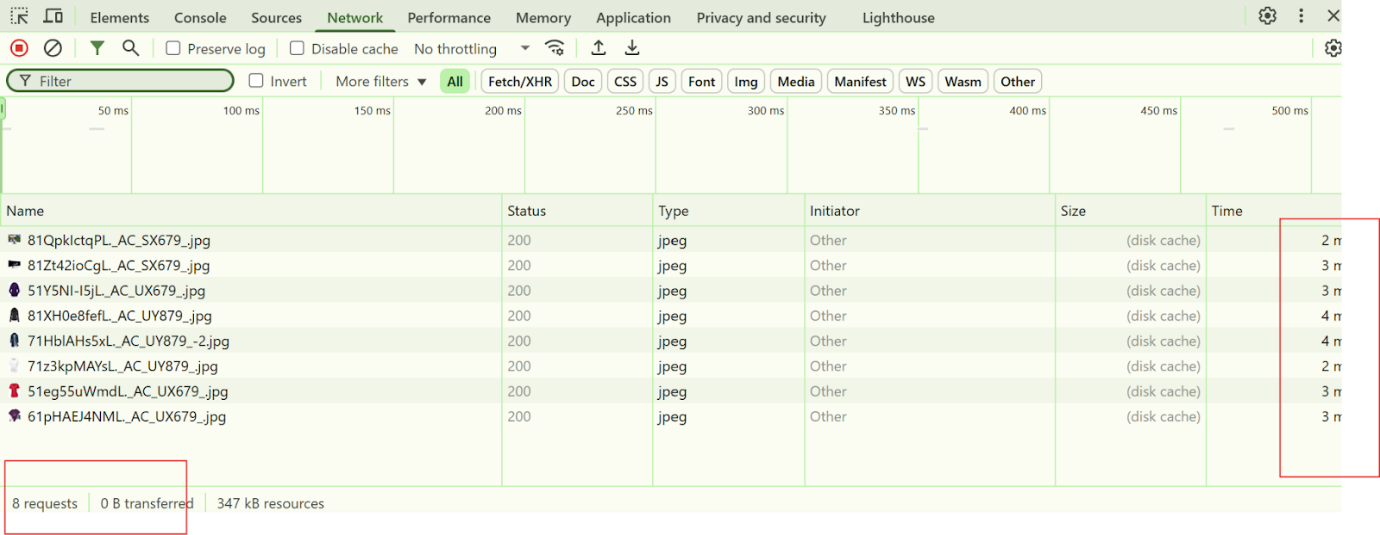
1. **Load Times**:
   * **Before**: 28.64ms
   * **After**: 20.59ms
   * **Improvement**: **8.05ms reduction** (28.1% improvement).
   * **Impact**: The application loads faster, providing a smoother experience for users.



1. **Render Performance (Longest Task)**:
   * **Before**: 750.0ms (Poor)
   * **After**: 83.3ms (Good)
   * **Improvement**: **666.7ms reduction** (88.9% improvement).
   * **Impact**: The main thread is no longer blocked for long periods, resulting in a more responsive application.



1. **Network Requests**:
   * **Before**: 8
   * **After**: 8
   * **Improvement**: No change.
   * **Impact**: The number of requests remains efficient.



1. **Server Bundle Size**:
   * **Before**: 597.69 kB
   * **After**: 597.69 kB
   * **Improvement**: No change.
   * **Impact**: The server bundle size remains large, indicating a need for further optimization.

**3. What Was Optimized**

1. **Compressed and Lazy Loaded Images**:
   * Images were compressed using tool called **iLoveImg**.
   * Lazy loading was implemented for offscreen images using the loading="lazy" attribute.
   * **Impact**: Reduced image load times and improved LCP.
2. **Implemented Caching**:
   * caching was implemented using **rxjs** to cache frequently requested data..
   * **Impact**: Reduced server response times and improved load times.

**4. Performance Tab Analysis**

The **Performance tab** in Chrome DevTools was used to validate improvements:

* **Before Optimization**:
  + Long tasks (e.g., 750.0ms) blocked the main thread, causing jank.
  + High LCP due to unoptimized images and render-blocking resources.
* **After Optimization**:
  + Long tasks were reduced to 83.3ms, improving responsiveness.
  + LCP improved significantly due to optimized images and lazy loading.

**5. Insights and Recommendations**

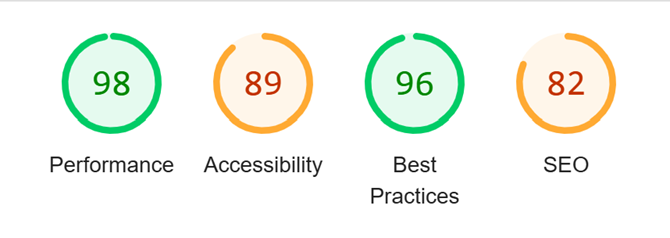
**Insights:**

* **Image Optimization**: Compressing and lazy loading images had the most significant impact on LCP.
* **Caching**: Implementing caching improved server response times and reduced load times.

**Recommendations:**

1. **Further Optimize Server Bundle**:
   * Use **tree shaking** and **code splitting** to reduce the server bundle size.
   * Remove unused dependencies and optimize third-party libraries.
2. **Reduce Network Requests**:
   * Combine and minify JavaScript/CSS files to reduce the number of requests.
3. **Monitor Performance**:
   * Use tools like **Lighthouse** and **Google Analytics** to monitor performance in production.
4. **Optimize Change Detection**:
   * Use Angular’s OnPush change detection strategy to further improve render performance.

**6. Summary of analysis lighthouse**



**7. Final Report Quality**

* **Clarity**: The report clearly explains the optimizations, metrics, and their impact.
* **Depth**: Detailed before-and-after comparisons provide a comprehensive analysis.
* **Completeness**: All key performance metrics (LCP, load times, render performance, etc.) are covered.
* **Structure**: The report is well-organized, with clear sections for metrics, optimizations, and recommendations.

**Conclusion**

The optimizations implemented (compressing and lazy loading images, implementing caching) resulted in **significant improvements** in LCP, load times, and render performance. However, further optimizations (e.g., reducing server bundle size, minimizing network requests) are recommended to achieve even better performance.