**Assignment 3 – Web Performance, Resilience & Security**

**Student: Yahya alsalmi - B00895877**Instructor: Dr. Oladapo Oyebode  
Course: CSCI 4177/5709 - Adv. Topics in Web Development - Summer 2025  
Date: 2025-07-27  
Repository: <https://github.com/EllE961/A3-CSCI4177.git>

# 1. Baseline Performance with JMeter

## 1.1 Scenarios

• Light: 10 users, 30s ramp-up.  
• Moderate: 50 users, 60s ramp-up; brief think time.

## 1.2 Results — Light Load (10 users, 30s ramp-up)

I included Aggregate Report (Light) CSV in the repository.

## 1.3 Results — Moderate Load (50 users, 60s ramp-up)

I included Aggregate Report (Moderate) CSV in the repository.

## 1.4 Top Bottlenecks (3)

1) GET /\_next/static/chunks/main-app.js?v=… (main JS bundle)  
 • Largest first-load asset; delays interactivity while downloading/parsing.  
 • Critical because it affects every user’s first paint and interactivity.  
 • Likely causes: limited code splitting, many eager client components, heavy above‑the‑fold work.

2) GET /api/products?page=1&limit=… (product list)  
 • High p95 under load; called frequently during browsing.  
 • Critical because it drives catalog browsing and perceived snappiness.  
 • Likely causes: no read‑through cache, large payloads (missing .select()/.lean()), few compound indexes.

3) GET /api/cart/totals (cart aggregation)  
 • Recalculated on every request; p95 degrades with concurrency.  
 • Critical on the checkout path; delays here raise drop‑off risk.  
 • Likely causes: repeated calculations without TTL cache; potential N+1 across services; no compression.

# 2. Client-Side Optimizations (Two)

## 2.1 Optimization #1 – React Query Caching & Request De‑duplication

Summary:  
Replace manual useEffect fetching on product list/detail pages with @tanstack/react-query so results are cached, concurrent requests are de‑duplicated, and back/forward navigation is instant.

Code/Config:  
• useQuery({ queryKey: ['products', query], queryFn: () => productService.getProducts(query), staleTime: 5–10 min }) on list page.  
• useQuery({ queryKey: ['product', id], queryFn: () => productService.getProductById(id), staleTime: 10 min }) on detail page.  
• No backend changes required.

Impact:  
• Fewer network calls on repeated views or shared data.  
• Lower perceived latency (cached data shown immediately).  
• Smoother transitions.

## 2.2 Optimization #2 – Image Optimization (responsive sizes, lazy, blur, careful priority)

Summary:  
Make product and hero images responsive and defer off‑screen images to improve LCP and reduce layout shifts.

Code/Config:  
• Add sizes="(max-width: 640px) 100vw, (max-width: 1024px) 50vw, 33vw" to grid images.  
• Add loading="lazy" to non‑critical images.  
• Add placeholder="blur" with a tiny blurDataURL.  
• Use priority only for above‑the‑fold hero images.

Impact:  
• Fewer bytes and less main‑thread work at first paint.  
• Faster LCP and better perceived speed, especially on mobile.

# 3. Server-Side Optimizations (Two)

## 3.1 Optimization #1 – Enable Response Compression (gzip/br)

Summary:  
Enable HTTP compression for JSON responses in services returning larger payloads (e.g., product, cart).

Code/Config:  
• import compression from 'compression'; app.use(compression({ threshold: 1024 })).

Impact:  
• 50–80% smaller payloads for list/detail JSON.  
• Lower average and p95 network time; more noticeable under moderate load.

## 3.2 Optimization #2 – Redis Read‑Through Cache for Product List (+ Invalidation)

Summary:  
Cache hot product list responses for 5 minutes using a key derived from query params; invalidate on create/update.

Code/Config:  
• Cache key example: products:${page}-${limit}-${sort}-${filter}.  
• On miss: query DB then cache.set(key, data, 300).  
• On write: cache.invalidate('products:\*') and cache.invalidate('product:<id>').

Impact:  
• Significant p95 reduction on warm traffic; higher throughput.  
• More stable latency under load.

# 4. Performance Comparison (Before vs After)

Attach JMeter Aggregate Reports (CSV) in the repo:

• Light Load — Before vs After (aggregate Light CSVs)  
• Moderate Load — Before vs After (aggregate Moderate CSVs)

# 5. Security: OWASP ZAP (Headless) & Remediations

## 5.1 Findings Summary (High/Medium)

• [10038] Content Security Policy (CSP) header not set / weak CSP (Client).  
• [10055] CSP weaknesses (API) due to missing explicit directives.  
• [10024] Sensitive information in URL (Auth) on /auth/login and /auth/register.

## 5.2 Remediation #1 – Strengthen Content Security Policy (CSP)

Before:  
Client pages and some static assets lacked CSP; API responses missed explicit directives.

Fix (code/config):  
• Next.js security headers for all routes plus /\_next/static/:path\*.  
• Helmet with explicit CSP in each Express service; disable X‑Powered‑By; strict referrer/cross‑origin policies.  
• Optionally add Cache‑Control: no‑store to sensitive endpoints; tighten CORS in dev.

After:  
CSP present across site and APIs; ZAP “CSP not set/weak” findings reduced/resolved (see ZAP after reports).

## 5.3 Remediation #2 – Prevent Credentials in URLs (Auth)

Before:  
Credentials were observed in query strings on auth pages.

Fix (code/config):  
• Next.js middleware strips sensitive params (email/password/etc.) on /auth/\* and redirects to clean URL.  
• API enforces POST‑only for /auth/login and /auth/register; added rate limiting and tighter CORS.

After:  
No credentials in URLs; GET on auth endpoints returns 405; ZAP alerts mitigated.

# 6. Monitoring: Prometheus + Grafana

6.1 Metrics & Scrape  
CPU, memory, request latency, error rate.

6.2 Dashboard Screenshots  
If dashboards fail to render, include prometheus.yml and docker compose configs; note blockers and next steps.

Status:  
I Spent significant time attempting setup but didn’t really work well. I Included Prometheus and Grafana configuration files in the repository for grading.

# Appendix – Artifacts Included in the repository

• JMeter: aggregate Light/Moderate CSVs (before/after)  
• ZAP reports: zap\_client\_before/after.html, zap\_api\_before/after.html  
• Monitoring configs: prometheus.yml, grafana provisioning  
• Updated code patches for security headers, middleware, Helmet, compression, Redis cache