Problem Statement: To enforce http3 usage, implement minification and Prevent browser debugging(Disable Devtools)

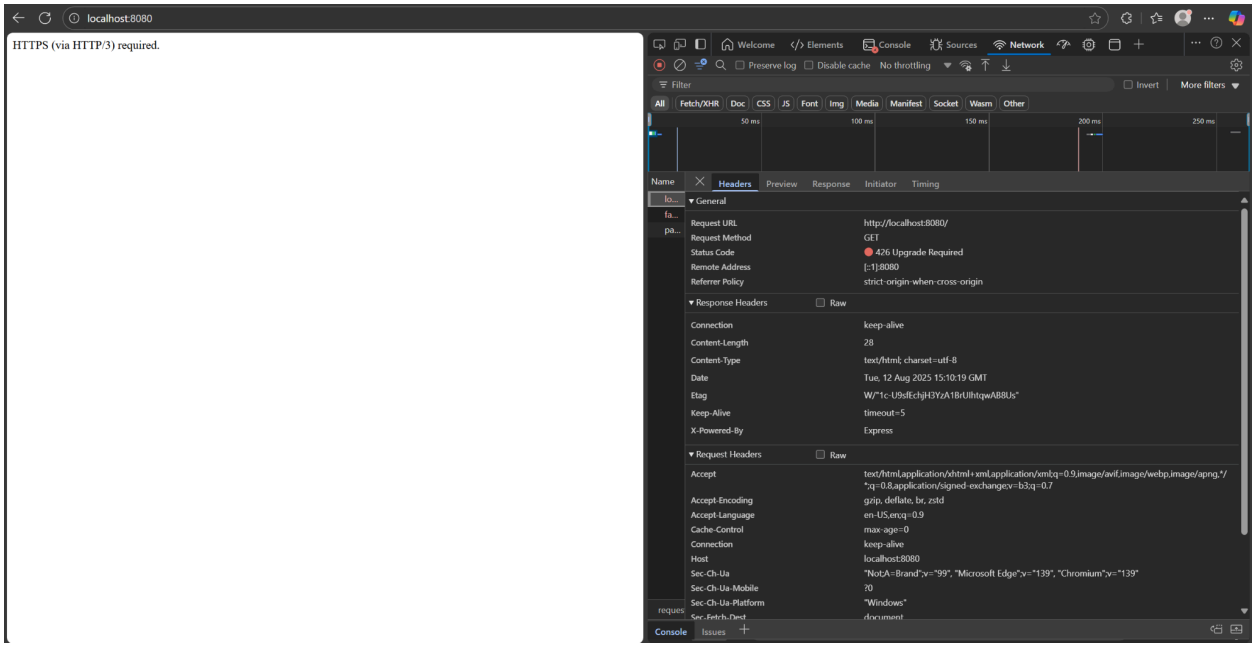
I have used a static html page as an example to test all the three tasks.

Github

Task 1: Enforce http3

We have used Node.js and caddy as a reverse proxy because it enables HTTP/3 support (not available in Node)

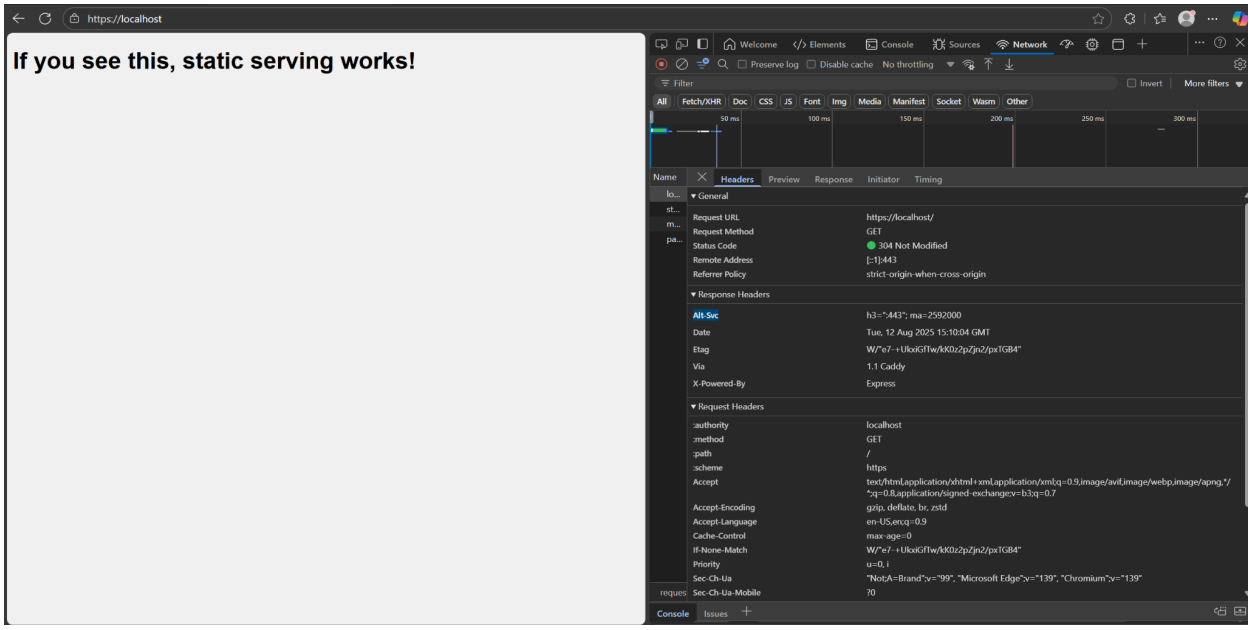
Without Caddy(non http3):
http://localhost:8080/
Status code: 426 is served here `'HTTPS (via HTTP/3) required.'`



Node is serving on port 8080 and caddy reverse proxies the same to port 433 for http3 support.

With Caddy(http3):

https://localhost/



Code:

```
function forceHTTPSOnly(req, res, next) {
  const proto = (req.headers['x-forwarded-proto'] || '').toLowerCase();
  if (proto === 'https') return next();
  return res.status(426).send('HTTPS (via HTTP/3) required.');
}
```

Caddyfile:

```
localhost:443 {
reverse_proxy localhost:8080 {
header_up x-forwarded-proto {scheme}
}
}
```

Task 2: Minification

Line wrap

```
1 <!doctype html><html><head><title>Test Page</title><link rel=stylesheet hr
```

Code:

```
async function minifyFile(filePath, ext) {
  const data = await fs.promises.readFile(filePath);
  const src = data.toString('utf8');

  if (ext === '.js') {
    const result = UglifyJS.minify(src);
    if (result.error) throw result.error;
    return result.code;
  } else if (ext === '.css') {
    const result = new CleanCSS().minify(src);
    if (result.errors && result.errors.length) throw new
Error(result.errors.join('\n'));
    return result.styles;
  } else if (ext === '.html') {
    const minified = await minifyHTML(src, {
      removeComments: true,
      collapseWhitespace: true,
      collapseInlineTagWhitespace: true,
      minifyCSS: true,
      minifyJS: true,
      removeAttributeQuotes: true,
      useShortDoctype: true,
    });
    // Inject obfuscated anti-debug script
    const injected = (minified.includes('</body>'))
      ? minified.replace('</body>', makeObfuscatedAntiDebug() + '</body>')
```

```
      : minified + makeObfuscatedAntiDebug();
    return injected;
  } else {
    return data; // Binary/other types untouched
  }
}
```

Once the minification was done knowing that it was a static website i was running at every point in time i ended up implementing caching too, How it helps us is we don't have to minify the website for every request and rather minify it once and send it at every request. The cached data is stored in RAM and is only available till the node server is up. When we restart the server, the very first request that comes in creates the cache again.

Cache code:

```
async function buildCache(publicDir) {
  const cache = new Map();

  async function walk(dir) {
    const entries = await fs.promises.readdir(dir, { withFileTypes: true });
    for (const ent of entries) {
      const full = path.join(dir, ent.name);
      if (ent.isDirectory()) {
        await walk(full);
      } else if (ent.isFile()) {
        const rel = '/' + path.relative(publicDir, full).replace(/\\/g, '/');
        const ext = path.extname(full).toLowerCase();
        try {
          const content = await minifyFile(full, ext);
          cache.set(rel, {
            content: Buffer.isBuffer(content) ? content : Buffer.from(content,
'utf8'),
            type: contentTypeFor(ext),
          });
        } catch (err) {
          const raw = await fs.promises.readFile(full);
          cache.set(rel, { content: raw, type: contentTypeFor(ext) });
          console.warn(`Minify failed for ${rel}, serving raw. Error:
${err.message}`);
```

```
        }
      }
    }
  }

  await walk(publicDir);
  return cache;
}

// --- Watch & Update Cache ---
function attachWatcher(publicDir, cache) {
  fs.watch(publicDir, { recursive: true }, async (eventType, filename) => {
    if (!filename) return;
    const full = path.join(publicDir, filename);
    const rel = '/' + filename.replace(/\\/g, '/');
    const ext = path.extname(full).toLowerCase();

    try {
      const stat = await fs.promises.stat(full);
      if (!stat.isFile()) return;
    } catch {
      cache.delete(rel);
      return;
    }

    try {
      const content = await minifyFile(full, ext);
      cache.set(rel, { content: Buffer.from(content, typeof content === 'string'
? 'utf8' : undefined), type: contentTypeFor(ext) });
    } catch (err) {
      const raw = await fs.promises.readFile(full);
      cache.set(rel, { content: raw, type: contentTypeFor(ext) });
      console.warn(`Minify failed for ${rel}, raw file used. Error:
${err.message}`);
    }
  });
}

// --- Middleware to Serve Cache ---
```

```
function cacheMiddleware(cache) {
  return function (req, res, next) {
    const url = req.path === '/' ? '/index.html' : req.path;
    if (cache.has(url)) {
      const entry = cache.get(url);
      res.type(entry.type).send(entry.content);
      return;
    }
    next();
  };
}
```

Task 3:Prevent Browser Debugging (Disable DevTools)

* Blocks F12, Ctrl+Shift+I/C/J
* Disables right-click
* Blocks Copy and Pasting content on the screen

Code:

```
function makeObfuscatedAntiDebug() {
  const raw = `
(function(){
  function blockKeys(e){
    if(e.key === 'F12' ||
       (e.ctrlKey && e.shiftKey && ['I','C','J'].includes(e.key)) ||
       (e.ctrlKey && e.key === 'U')){
      e.preventDefault();
      return false;
    }
  }
  function blockContext(e){ e.preventDefault(); return false; }
  function blockSelect(){ return false; }
  function blockDrag(e){ e.preventDefault(); return false; }
  function blockCopy(e){ e.preventDefault(); return false; }
  function blockCut(e){ e.preventDefault(); return false; }
  function blockPaste(e){ e.preventDefault(); return false; }

  document.addEventListener('keydown', blockKeys, true);
  document.addEventListener('contextmenu', blockContext, true);
  document.addEventListener('selectstart', blockSelect, true);
  document.addEventListener('dragstart', blockDrag, true);
  document.addEventListener('copy', blockCopy, true);
  document.addEventListener('cut', blockCut, true);
  document.addEventListener('paste', blockPaste, true);

  setInterval(function(){
    try {
      if(window.outerWidth - window.innerWidth > 100 ||
         window.outerHeight - window.innerHeight > 100){
```

```
        document.documentElement.innerHTML = '<h1>DevTools Disabled</h1>';
      }
    } catch(e){}
  }, 1000);
})();`;

  // Obfuscate into String.fromCharCode
  const codes = Array.from(raw).map(c => c.charCodeAt(0));
  return `<script>eval(String.fromCharCode(${codes.join(',')}));</script>`;
}
```