# Week 11 Technical Task Report – Evidence & Reflection

*Unit: SIT314 / Module Pass Task*

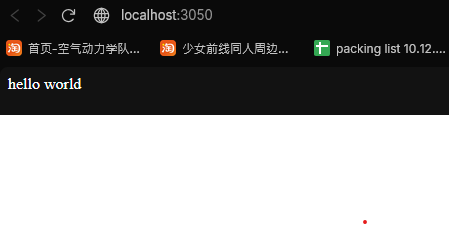
## Overview

This section documents the security improvements applied to a Node.js (Express) web server for IoT use. The evidence follows the exact order from the task instructions: (1) Baseline HTTP, (2) Baseline headers before Helmet, (3) HTTPS with self‑signed certificate, and (4) Headers after enabling Helmet. It then provides a brief step‑by‑step summary and a reflection on what was learned.

## Evidence (ordered by the instructions)

### Screenshot 1 — Baseline HTTP server running (server.js)

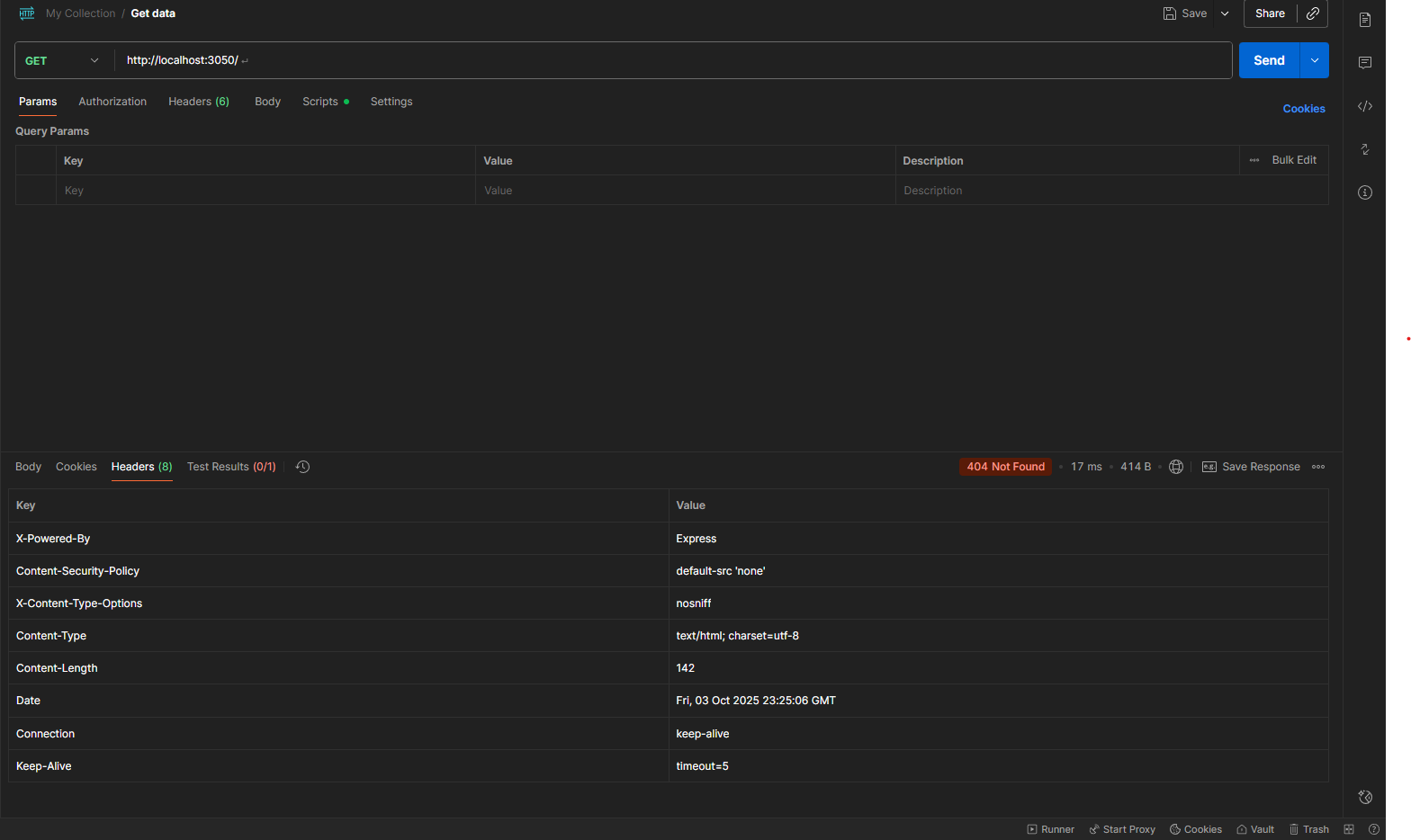
What it shows: The plain Express server is reachable at http://localhost:3050/ and returns “hello world”. This establishes the unsecured baseline (no TLS). Why it matters: Confirms the app works before adding security layers.



### Screenshot 2 — Baseline headers BEFORE Helmet (Postman)

What it shows:

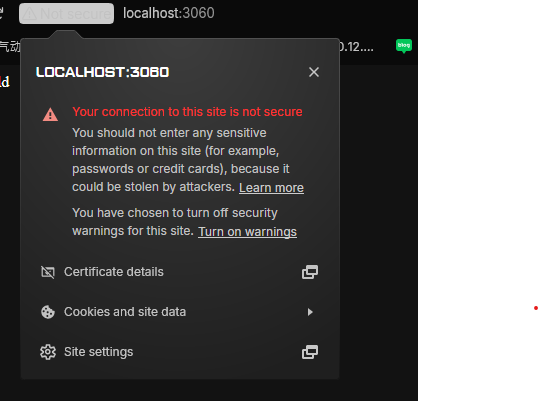
A GET to http://localhost:3050/ returns minimal headers (e.g., X‑Powered‑By: Express) and lacks common security headers such as X‑Content‑Type‑Options, X‑Frame‑Options, Referrer‑Policy, and a strict Content‑Security‑Policy (CSP). Why it matters: Captures the “before” state to compare the effect of Helmet later. Note: The 404 status in the screenshot is acceptable for demonstrating headers—the header set is what we compare.



### Screenshot 3 — HTTPS with self‑signed certificate (serversecure.js)

What it shows:

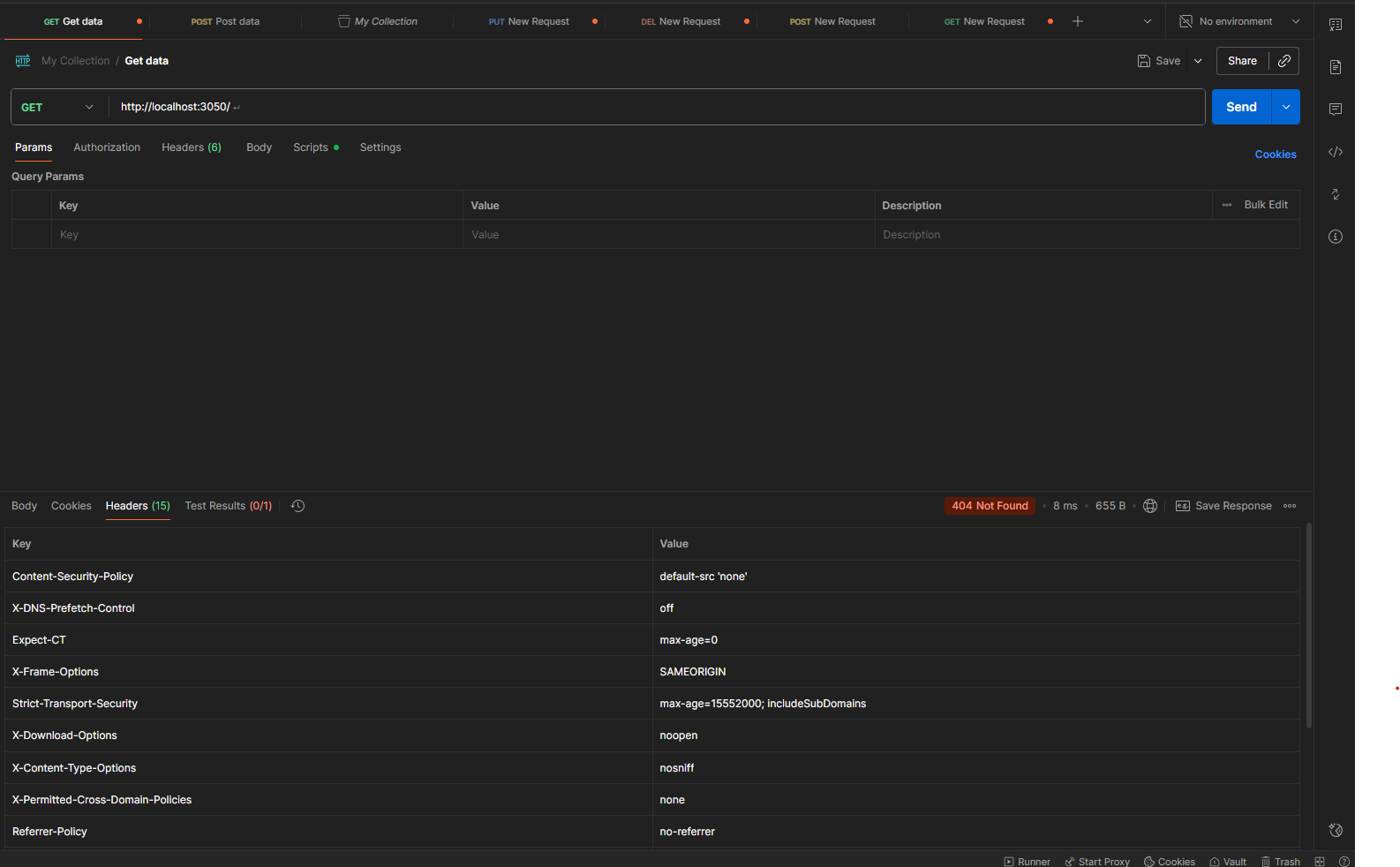
Visiting https://localhost:3060/ prompts a browser warning (“connection is not secure”), which is expected with a self‑signed certificate in development. Why it matters: Proves the app now serves over HTTPS; production should use a CA‑signed certificate so browsers trust it automatically.



### Screenshot 4 — Headers AFTER enabling Helmet (Postman)

What it shows:

A GET to http://localhost:3050/ now includes multiple defensive security headers, e.g., Content‑Security‑Policy (locks down resources), X‑Frame‑Options: SAMEORIGIN (mitigates clickjacking), X‑Content‑Type‑Options: nosniff (prevents MIME sniffing), Referrer‑Policy (reduces data leakage), X‑DNS‑Prefetch‑Control, X‑Permitted‑Cross‑Domain‑Policies, and Strict‑Transport‑Security (effective over HTTPS). Why it matters: Demonstrates concrete hardening from Helmet compared to the baseline.



## Step‑by‑Step (Brief)

• Baseline HTTP: Created server.js (Express), started on port 3050, verified “hello world” in the browser (Screenshot 1).

• Headers (before): Queried http://localhost:3050/ in Postman to capture baseline headers without Helmet (Screenshot 2).

• Enable HTTPS: Generated self‑signed certs with OpenSSL, created serversecure.js, served on port 3060; acknowledged the self‑signed browser warning (Screenshot 3).

• Helmet hardening: Installed and conScreenshotd Helmet; re‑queried the server and verified additional security headers now appear (Screenshot 4).

## Reflection

Defense‑in‑depth: Moving from plain HTTP to HTTPS protects confidentiality and integrity of traffic, while Helmet adds layered mitigations against common web risks (XSS, clickjacking, MIME sniffing).

Dev vs. Prod: Self‑signed certificates are suitable for local testing but cause warnings; production requires a CA‑signed certificate and enables HSTS safely over HTTPS.

Evidence‑driven verification: Comparing headers before vs. after made security improvements tangible, independent of response status codes or body content.

Gotchas resolved: Quoted Windows paths for folders with spaces, added OpenSSL to PATH, and ensured Helmet version compatibility (pin helmet@5 for per‑middleware API).