📄 WEEK 1 — ENGINEERING MINDSET BOOTCAMP

Objective: Train interns to think like engineers — not just code, but research, debug, measure, document, and deliver.

DAY 1 — SYSTEM REVERSE ENGINEERING + NODE & TERMINAL MASTERING

🔹 Learning Outcomes

- Master terminal navigation and system inspection
- Deep understanding of PATH, environment variables, Node runtime

🔹 Tasks (NO GUI allowed — only terminal)

1. Identify and document:
    - OS version
    - Current shell (`bash/zsh/powershell`)
    - Node binary path (`which node`)
    - NPM global installation path
    - All PATH entries that include "node" or "npm"
2. Install & use **NVM** (Node Version Manager)
    - Install NVM
    - Switch Node from LTS → Latest and back
3. Create script `introspect.js` that prints:

```
OS:
Architecture:
CPU Cores:
Total Memory:
System Uptime:
Current Logged User:
Node Path:
```

4. STREAM vs BUFFER exercise (performance benchmark)
    - Create a large test file (50MB+)
    - Read file using both:
        - fs.readFile (Buffer)
        - Stream (fs.createReadStream)
    - Capture execution time + memory usage

🔹 Deliverables

Deliverable	Format
system-report.md	Document with screenshots
introspect.js	JS script
logs/day1-perf.json	Execution time + memory usage
commits	(Minimum 6 commits with meaningful messages)

DAY 2 — NODE CLI APP + CONCURRENCY + LARGE DATA PROCESSING

🔹 Learning Outcomes

- Asynchronous programming
- CLI tool building
- Concurrency + performance measurement

🔹 Tasks

1. Generate a **corpus text file** with 200,000+ words (random lorem or internet scrape)
2. Build CLI command:

```
node wordstat.js --file corpus.txt --top 10 --minLen 5 --unique
```

3. The CLI must output:
    - Total words
    - Unique words
    - Longest word

    - Shortest word
    - Top N most repeated words
4. Implement concurrency:
    - Divide file into chunks
    - Process chunks in parallel using Promise.all or worker_threads
5. Benchmark performance for concurrency levels:
    - Concurrency 1, 4, 8
    - Capture run performance in logs

🔹 Deliverables

Deliverable	Format
wordstat.js	Executable CLI tool
output/stats.json	Final computed results
logs/perf-summary.json	Concurrency test with runtime
commits	Minimum 8 commits documenting progress

DAY 3 — GIT MASTERY: RESET, REVERT, CHERRY-PICK, BISECT, STASH

🔹 Learning Outcomes

- Ability to **recover from mistakes**
- Proper commit discipline

🔹 Tasks

1. Create repository with 8+ commits
    - intentionally introduce a bug in commit 4
2. Use `git bisect` to detect the faulty commit
3. Fix bug, then `git revert` (not reset) only the buggy commit
4. Use stash workflow:

```
git stash
git pull
git stash apply
```

5. Using two clones of the same repo:
    - Edit the same line in same file
    - Merge and resolve conflict (must keep both changes)

🔹 Deliverables

Deliverable	Format
bisect-session.txt	Terminal log
stash-session.txt	how stash fixed workflow
MERGE-POSTMORTEM.md	Explanation screenshot + resolution
commits	Graph must show branches + merge

DAY 4 — HTTP / API FORENSICS (USING CURL + POSTMAN + HEADERS)

🔹 Learning Outcomes

- Headers
- Pagination
- ETag caching
- Understanding request–response cycle

🔹 Tasks

Perform DNS lookup and traceroute:
nslookup dummyjson.com
traceroute dummyjson.com

1.

Using CURL:
curl -v https://dummyjson.com/products?limit=5&skip=10

2.
3. Modify headers:
    - Remove `User-Agent`

    - Send fake Authorization header
    - Capture differences
4. Observe caching:
    - Get response ETag

Re-send request using:

```
curl -H "If-None-Match: <etag>"
```

    - Expect 304 (Not Modified)
5. Build a small Node HTTP server with endpoints:
    - `/echo` → return headers
    - `/slow?ms=3000` → delay response by query param
    - `/cache` → return cache headers

🔹 Deliverables

Deliverable	Format
curl-lab.txt	Requests + responses
api-investigation.md	Analysis (pagination + headers + caching)
server.js	Node server
screenshots	For POSTMAN requests

DAY 5 — AUTOMATION & MINI-CI PIPELINE

🔹 Learning Outcomes

- Automation mindset
- Build safeguards to prevent bad commits

🔹 Tasks

1. Create `validate.sh` script:
    - Ensure `src/` exists
    - Ensure config.json is valid

    - Append logs with timestamps
2. Add ESLint + Prettier:
    - Bad formatting must block commit
3. Add pre-commit hook using husky:
    - Runs lint, validate.sh
    - Reject commit if script fails

Create build artifact:
build-<timestamp>.tgz

4.
    - Include logs, source code
    - Generate SHA checksum
5. Schedule script execution:
    - cron (Linux/Mac) or Task Scheduler (Windows)

🔹 Deliverables

Deliverable	Format
validate.sh	Must exit non-zero on error
.eslintrc + Prettier config	Required
husky hook screenshot	Must show failed commit
artifacts/build-*.tgz	Evidence of packaging
WEEK1-RETRO.md	Lessons learned & what broke