

Node.js Backend Security - Foundations

Visual handwritten workbook - Tier 1

Understand

Clear explanations and real failure stories

Visualize

Diagrams that expose trust boundaries

Practice

Original exercises and runnable tests

Remember

Quizzes, solutions and review checklists

19 topics - original TeamNotes labs - explained answer key

A digital asset by Sooraj | soorajdev.cloud

Expanded edition 2.0 | Reviewed 25 October 2026

Course map I

Topic IDs match the companion tests across both tiers.

Topic	Concept / lab / solution
01 Think in trust boundaries	6 / 7 / 44
19 One request, one meaning	8 / 9 / 45
02 Configuration that fails closed	10 / 11 / 46
03 Validate shape, type and meaning	12 / 13 / 47
20 Render data as data	14 / 15 / 48
21 NoSQL still has query structure	16 / 17 / 49
06 Queries that keep input as data	18 / 19 / 50
04 Authorise the object, every time	20 / 21 / 51
22 Roles, actions and tenants	22 / 23 / 52
05 Control writes and responses	24 / 25 / 53

Follow the learning path

Topic numbers remain stable across editions so they match tests and the repository crosswalk. Follow the order in this map rather than sorting by number. Every topic has an explanation, an exercise and an explained solution.

TRY FIRST. Quizzes show questions only. Their answers are collected in the final section, after the solutions, reference directory and repository coverage map.

Course map 2

Topic IDs match the companion tests across both tiers.

Topic	Concept / lab / solution
07 Passwords without shortcuts	26 / 27 / 54
08 Sessions with a lifecycle	28 / 29 / 55
09 CSRF, origins and CORS	30 / 31 / 56
23 Transport, headers and redirects	32 / 33 / 57
24 Encrypt with context and integrity	34 / 35 / 58
25 Own asynchronous failures	36 / 37 / 59
26 Give each process less authority	38 / 39 / 60
27 Know the API you expose	40 / 41 / 61
28 Protect the business transition	42 / 43 / 62

What this tier gives you

Build a defensible private-note API boundary: valid input, correct ownership, safe fields and queries, password verification, sessions and CSRF. Finish with a route-review assessment that connects these controls.

START HERE. You should know JavaScript functions, objects, promises and basic HTTP. You do not need earlier security training.

How to use the answer key

Write the expected result before running a test. Attempt the implementation in `exercises/security.mjs`. Read the solution only after you can explain the failing case. The full source in `solutions/security.mjs` includes shared imports that shortened printed snippets assume.

After the topic solutions: final assessment, version and scope notes, reference directory, repository coverage map, then the separate quiz answer key. PDF bookmarks navigate all sections.

Your local lab

One small, dependency-free project accompanies both tiers.

1. Extract the companion

Extract Nodejs_Security_Companion.zip into a private working folder. Open a terminal in the extracted folder containing package.json. Commands below work in PowerShell, Terminal and most shells.

```
node --version
npm test
# Exercise tests intentionally fail until implemented:
npm run test:exercises
```

4 lines

2. Work one topic at a time

```
node --test --test-name-pattern L03 tests/exercises.test.mjs
node --test --test-name-pattern L03 tests/solutions.test.mjs
```

2 lines

For IDs 19-39, use tests/expanded-exercises.test.mjs and tests/expanded.test.mjs instead. Topics 01 and 17 are design exercises. Topic 18 is the HTTP capstone. Filtered-out tests are skipped normally. No npm install is needed.

3. Keep roles clear

File	Purpose
exercises/security.mjs + expanded.mjs	Your implementations; TODO skeletons
solutions/security.mjs + expanded.mjs	Complete reference implementations
tests/suite.mjs + expanded-suite.mjs	Shared behavioural assertions
server.mjs	Local integrated solution, not your exercise file
README.md	HTTP walkthrough, scope and troubleshooting

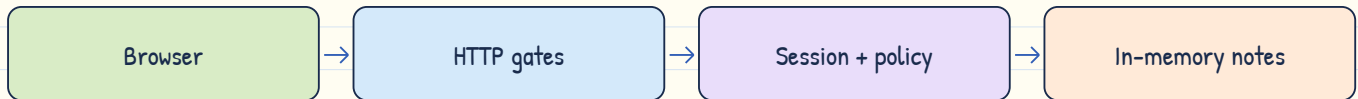
IMPORTANT. The server imports solutions, not your exercise skeleton. Passing the starter tests proves your helpers; npm start demonstrates the supplied complete application.

Meet TeamNotes

Asha and Ravi are fictional users. All credentials and notes are study fixtures.

TeamNotes is a deliberately small backend for private note titles. Each note has an owner. The browser signs in, receives a cookie, retrieves its session token and requests a specific note. This gives us a concrete policy to test repeatedly without pretending to build a complete SaaS product.

DIAGRAM / FOLLOW THE BOUNDARY



Fixture	Value
Asha	username asha; userId u1; note 1
Ravi	username ravi; userId u2; note 2
Study password	study-only-passphrase
Local address	http://127.0.0.1:3000
Session policy	30-minute absolute expiry; rotation; logout

The boundary contract

- + Anonymous callers cannot read private notes. A signed-in caller can read and update only their own note.
- + The caller can change title only. Internal fields never enter the public note response.
- + Browser writes require an exact Origin, JSON content type and a session CSRF token. Login has an Origin check before a session exists.
- + Request bytes, attempts and concurrent password work are bounded in the local demonstration.

LAB SCOPE. Run only the supplied loopback lab and your own test fixtures. The vulnerable snippets are explanations; the exercise skeleton throws TODO errors rather than exposing a vulnerable service.

What is not wired into the server

SQLite search, password reset, outbound URL construction, uploads, child-process export and webhook verification are isolated tested modules. There are no hidden endpoints, email senders, payment integrations or production deployment scripts.

Think in trust boundaries

Map what a caller controls before choosing a security library.

THE FAILURE STORY. TeamNotes lets Asha and Ravi keep private project notes. Asha signs in and asks for note 2. The ID is valid, but note 2 belongs to Ravi. A response can be perfectly formatted and still expose the wrong person's data.

A trust boundary is a place where data moves between components with different authority. The browser can suggest a note ID, title or file name. It cannot decide the authenticated user, ownership, database role or destination network. Treat those decisions as server responsibilities.

Start with the valuable things: private note text, credentials, session tokens and the service's availability. Then describe a misuse in one sentence: a signed-in user reads a different owner's note. Link that misuse to a control and an observable test. This makes a threat model useful during a code review.

Use four questions for every route: who is calling, what may they act on, which fields may change, and what work can this request trigger? Authentication answers only the first. Secure behaviour comes from answering all four, including when a request is rejected.

DIAGRAM / FOLLOW THE BOUNDARY



Caller sends	Server establishes
noteId = 2	ownership from stored data
ownerId = u1	userId from authenticated session
role = admin	privileges from trusted account state
X-Forwarded-For	peer identity from deployment policy

Spot the mistake

UNSAFE / INCOMPLETE EXAMPLE

3 lines

```
const note = notes.find(n => n.id === req.params.id);
// Being signed in does not prove ownership.
res.json(note);
```

WATCH OUT. A random-looking ID reduces guessing; it does not grant permission. A hidden button is not an access-control boundary.

Lab: Think in trust boundaries

YOUR TASK. Create a one-page threat model for TeamNotes before editing code.

Work through it

1. List the browser, Node process, session store and note store. Draw arrows for login, read and update.
2. For each arrow, mark which fields are untrusted. Include cookies and headers; do not treat a signed-in browser as trusted input.
3. Write three misuse stories: cross-owner reading, owner reassignment and repeated expensive login attempts.
4. For each story, record a control, a negative test and one remaining risk. Give the risk an owner rather than writing simply "secure".

Acceptance cases

Case	Expected behaviour
Asha reads note 1	Allowed; only public response fields
Asha reads note 2	Rejected before note data leaves the service
Asha submits ownerId	Rejected; stored owner unchanged
Many expensive logins	Bounded attempts and bounded concurrent hashing

Mini quiz

Q1. Can a valid request be unauthorised?

Q2. Is every caller malicious?

Try both questions before checking the quiz answer key at the end of this workbook.

Key takeaways

- + Name the asset, misuse, control and test together.
- + Keep identity and resource ownership separate.
- + Reject by policy before exposing or changing data.

STRETCH. Add a support role. Decide whether support can read note contents, and how that exceptional access would be recorded.

Further reading: R1, R2, R3. Full links appear in the reference directory.