

Node.js Backend Security - Advanced

Visual handwritten workbook - Tier 2

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

20 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
10 Recovery tokens are credentials	6 / 7 / 46
29 JWT, OAuth and OIDC boundaries	8 / 9 / 47
11 Avoid arbitrary outbound URLs	10 / 11 / 48
34 Distrust upstream responses too	12 / 13 / 49
12 Files need more than an extension	14 / 15 / 50
35 Archives, streams and parser limits	16 / 17 / 51
13 Run programs without a shell	18 / 19 / 52
30 Prototype pollution and code input	20 / 21 / 53
31 Keep the event loop available	22 / 23 / 54
14 Webhooks: bytes, time and replay	24 / 25 / 55

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
32 One effect under concurrent requests	26 / 27 / 56
15 Budget bytes, attempts and work	28 / 29 / 57
16 Useful errors, minimal logs	30 / 31 / 58
33 Dependencies and secure delivery	32 / 33 / 59
17 Harden the release path	34 / 35 / 60
36 Rotate, revoke and recover	36 / 37 / 61
37 Real-time channels need policy	38 / 39 / 62
39 AI tools inherit no authority	40 / 41 / 63
38 Review with repeatable evidence	42 / 43 / 64
18 Capstone: review the whole flow	44 / 45 / 65

What this tier gives you

Extend a secure foundation into recovery, network and filesystem boundaries, process execution, webhook integrity and operational hardening. Finish by reviewing the full HTTP application and its production gaps.

BEFORE YOU BEGIN. You should already understand validation, ownership, password hashing, cookies and CSRF. The companion includes all solutions, so you can inspect prerequisites without buying another file.

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

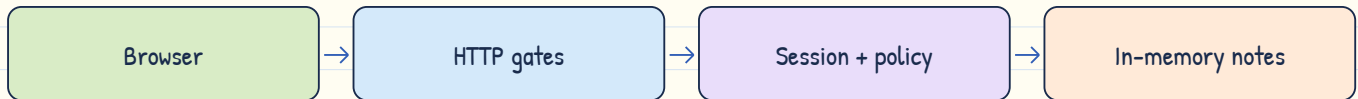
IMPORTANT. The server imports solutions, not your exercise skeleton. Passing the starter tests proves your helpers; npm start demonstrates the supplied complete application.

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.

Recovery tokens are credentials

Make password reset short-lived, single-use and consistent under races.

THE FAILURE STORY. A password-reset link is used once successfully and then used again. If the token remains valid, anyone holding an old email can change the password later. Two simultaneous requests also need one consistent winner.

Recovery bypasses the usual password check, so its token is a powerful credential. Generate a high-entropy random value, store only its digest, bind it to one account and set a short expiry. A fast hash is appropriate for this random token; it is not appropriate for a human-chosen password.

The local `ResetTokens` class invalidates earlier tokens for the same user and consumes a matching token synchronously. This gives single-use behaviour within one JavaScript process. A multi-instance database implementation must consume the token and update the password in one atomic transaction.

The request-reset endpoint should return a generic acknowledgement whether an account exists or not. Send links using a fixed configured origin, not the Host header supplied by a caller. After reset, revoke existing sessions and notify the account owner through the product's verified channel.

DIAGRAM / FOLLOW THE BOUNDARY



Property	Reason
Digest stored	Database read does not reveal raw links
Short expiry	Limits exposure lifetime
One use	Old links cannot keep changing credentials
Atomic update	Two workers cannot both complete reset

Spot the mistake

UNSAFE / INCOMPLETE EXAMPLE 3 lines

```

const userId = resetTokens.get(token);
await updatePassword(userId, newPassword);
// Token is still reusable and races are unspecified.
  
```

WATCH OUT. Do not consume a token, then perform an unrelated database update without a recovery plan. A failed password update must not create an inconsistent recovery state.

Lab: Recovery tokens are credentials

YOUR TASK. Implement `ResetTokens`. The exercise models token lifecycle only; it does not send email or change real passwords.

Work through it

1. Generate 32 random bytes and return a hex token. Store a SHA-256 digest as the Map key.
2. Invalidate earlier tokens for the same user and prune expired entries. Keep the demonstration store bounded.
3. Consume only correctly shaped, unexpired tokens. Delete the stored entry before returning the `userId` in this synchronous model.
4. Run L10 tests. Add a test showing a token for `u1` cannot consume the token for `u2`.

Acceptance cases

Case	Expected behaviour
First use before expiry	Returns bound <code>userId</code>
Second use	Rejected
Use at exact expiry	Rejected
New token for same account	Earlier token rejected

Mini quiz

Q1. Why SHA-256 here but not for passwords?

Q2. Does deleting a Map entry prove distributed safety?

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

Key takeaways

- + Treat reset links as bearer credentials.
- + Store a digest, expiry and account binding.
- + Make consumption and credential change atomic in production.

STRETCH. Write a database transaction outline that atomically marks a reset token used, changes the password and revokes sessions.

Further reading: R15, R13. Full links appear in the reference directory.