CODE REVIEW EVALUATION FORM

JavaScript & Express.js | Undergraduate Programming Course

1. SUBMISSION INFORMATION

Course:	**ICS 385**	**Section:**	**1**
Instructor:	**Debasis Bhattacharya**	**Semester:**	**Spring 2026**
Student Name:	**Jonas Marcial**	**Student ID:**	**26502437**
Project Title:	**Week 5 Code Review**	**Date:**	**2/13/26**
Reviewer:	**Jonas Marcial**	**Review Type:**	Peer / Instructor

2. CODE SUBMISSION DETAILS

Repository URL:	**https://github.com/jmarcial2001/ics385spring2026.git**		
Branch:	**main**	**Commit Hash:**	
Files Reviewed:	**solution.js**	**Lines of Code:**	**35 lines**

3. CODE OVERVIEW & PURPOSE

Briefly describe the purpose of the submitted code, its main functionality, the Express.js routes implemented, and any middleware or external packages used.

Summary: The purpose of this project is to create a simple Express.js app that asks the user for a password and shows a secret page if the password is correct. The app uses Express routes and middleware to read form input - req.body.password.

It includes a get route (/) to show the password form, and a post route (/check) to either show the secret page or reload the form. The app uses body-parser middleware for parsing form data and a custom middleware function, passwordCheck, to check the password.

4. EVALUATION CRITERIA

Rate each criterion on the scale provided. Use the descriptors as guidance. A score of 4 = Excellent, 3 = Proficient, 2 = Developing, 1 = Beginning, 0 = Not Attempted.

Code Correctness & Functionality	Runs and works as expected (loads form, checks password, shows secret). Doesn’t handle edge cases beyond basic flow.	3	20%

Code Structure & Organization	Everything is in one file; no separation into routes/controllers. Fine for a small demo but not scalable.	2	15%
Naming Conventions & Readability	Readable names overall (passwordCheck, userIsAuthorised). Code is easy to follow.	3	10%
Express.js Best Practices	basic middleware + routes are correct, but no Router structure, no proper auth/session handling, and middleware is applied globally in a way that’s not ideal.	2	15%
Error Handling & Validation	No real validation, no try/catch, no error responses, no helpful feedback to user.	1	10%
Comments & Documentation	Some comments exist, but not much explanation of security or design decisions.	2	10%
Security Considerations	Hardcoded password, global auth flag, no sessions, no rate limiting, and secret is a static HTML file.	1	10%
Testing & Reliability		1	10%

Total Weighted Score:	2 / 4.00	**Percentage:**	70%

5. DETAILED FINDINGS — CODE-LEVEL OBSERVATIONS

Document specific issues, bugs, or noteworthy patterns found during the review. Reference file names and line numbers where applicable.

1	Original main server file	**High** / Med / Low	Security	Uses a global variable userIsAuthorised. Once set to true, it applies to everyone using the server until restart.
2	(Original main server file)	**High** / Med / Low	Security	Password is hardcoded in the source ("ILoveProgramming"). Anyone with the repo can see it.
3	POST handler	High / **Med** / Low	Validation	app.use(passwordCheck) runs on every request, including GET /. It relies on req.body, which isn't meaningful for GET requests.
4	POST handler response	High / **Med** / Low	Express Best Practices	No session/cookie authentication. Authorization is not tied to a specific user.
5	Overall app	High / **Med** / Low	Error Handling	If password is wrong, it silently reloads the index page. No message and no status code differences.
6	Form route / POST route	High / **Med** / Low	Security	Secret content is stored as a static HTML file. If the app were deployed incorrectly, this file could be accessed directly.
7	Any logging/console output	High / **Med** / Low	Security / Privacy	No rate limiting or brute-force prevention on /check
8	Project structure	High / Med / **Low**	Organization	Port is hardcoded to 3000 instead of using

6. EXPRESS.JS & JAVASCRIPT CHECKLIST

Check each item that applies to the submitted code. Mark Y (Yes), N (No), or N/A.

Server Setup	Server listens on a configurable port (e.g., process.env.PORT)	N
Server Setup	Entry point file is clearly identified (e.g., app.js or server.js)	Y
Routing	Routes are organized using express.Router()	N
Routing	RESTful conventions followed (GET, POST, PUT/PATCH, DELETE)	Y
Routing	Route parameters and query strings used correctly	N/A
Middleware	Body-parser or express.json() configured for request parsing	Y
Middleware	Custom middleware is reusable and well-documented	N

Middleware	Error-handling middleware defined with (err, req, res, next) signature	N
Async/Await	Promises and async/await used correctly (no unhandled rejections)	N/A
Async/Await	Callback patterns avoided in favor of modern async patterns	Y
Dependencies	package.json lists all dependencies; no unused packages	Y
Dependencies	node_modules excluded via .gitignore	Y

Security	Environment variables managed via .env / dotenv	N
Security	No sensitive data committed to version control	N

7. QUALITATIVE FEEDBACK

Strengths — What does this submission do well?

- The app works correctly and demonstrates the basics of Express routing, middleware, and handling form data. The code is short and easy to read, which makes it good for learning how requests flow through middleware into routes.

Areas for Improvement — What should the student focus on next?

Security and authentication need the most improvement.

The password should not be hardcoded, and authorization should not

be stored in a global variable.

The app should use sessions so login is per user, and it should add

rate limiting to reduce password guessing.

Error handling and user feedback could also be improved.

Suggested Learning Resources

:

8. OVERALL ASSESSMENT

A / Excellent	90–100%	Code is well-structured, fully functional, secure, and demonstrates mastery of Express.js concepts.
B / Proficient	80–89%	Code works correctly with minor issues; good organization and documentation; some improvements possible.
C / Developing	70–79%	Code runs but has notable gaps in structure, error handling, or best practices; needs revision.
D / Beginning	60–69%	Significant issues with functionality, structure, or documentation; substantial rework required.
F / Incomplete	Below 60%	Code does not compile/run or is largely incomplete; fundamental concepts not demonstrated.

Final Grade Assigned:	**C**	**Numeric Score: 74**	74/ 100

9. REQUIRED REVISIONS & ACTION ITEMS

List any mandatory changes the student must complete before resubmission.

1	Move password/secret to environment variables (.env) and remove hardcoded secret	**High** / Med / Low	
2	Replace global userIsAuthorised with session-based authentication	**High** / Med / Low	
3	Add rate limiting to /check to reduce brute force attempts Priority: Med	High / **Med** / Low	
4	Add basic validation/error feedback (wrong password message + status codes)	High / **Med** / Low	

10. ACADEMIC INTEGRITY ACKNOWLEDGMENT

By signing below, the reviewer confirms that this evaluation was conducted fairly and objectively. The student acknowledges receipt of this feedback and understands the revisions required.

Reviewer Signature:	**Jonas Marcial**	**Date:**	**2/14/26**
Student Signature:		**Date:**	

Instructor Signature:		**Date:**	