CODE REVIEW EVALUATION FORM

JavaScript & Express.js | Undergraduate Programming Course

1. SUBMISSION INFORMATION

Course:	**ICS-385-0: Web Devl-Admin**	**Section:**	**47033**
Instructor:	**Dr. Debasis Bhattacharya**	**Semester:**	**SP26**
Student Name:	*Udemy*	**Student ID:**	*n/a*
Project Title:	**3.5 Secrets Project**	**Date:**	**2/14/26**
Reviewer:	**Kendall Beam**	**Review Type:**	[✓ Peer] / Instructor

2. CODE SUBMISSION DETAILS

Repository URL:	*n/a*		
Branch:	*n/a*	**Commit Hash:**	*n/a*
Files Reviewed:	**solution.js, /3.5 Secrets Project**	**Lines of Code:**	**38**

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: simple webpage html form that loads a "secret" page if correct

- **uses express for webpage engine**
- **middleware:**
    - **bodyparser -> for parsing html form**
    - **(custom) passwordCheck -> if password is correct then set user auth variable**
    - **app.get -> send index.html**
    - **app.post -> if authorized send secret.html**
        - **else redirect/send back to index.html**
- **runs on localhost:3000**

4. 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.

Criterion	Description	Score (0–4)	Weight
Code Correctness & Functionality	Application runs without errors; all Express routes return expected responses; edge cases handled. - though missing a global route catch to 404?	4	20%

Criterion	Description	Score (0–4)	Weight
Code Structure & Organization	Logical file/folder structure (e.g., routes/, controllers/, models/); separation of concerns; modular design.	4	15%
Naming Conventions & Readability	Variables, functions, and routes use clear, descriptive names following camelCase conventions; consistent formatting.	4	10%
Express.js Best Practices	Proper use of Router, middleware chaining, error-handling middleware, appropriate HTTP methods and status codes.	4	15%
Error Handling & Validation	Input validation present; try/catch or .catch() used; meaningful error messages returned to client. - minimal validation through bodyParser - no error/validation messages	2	10%
Comments & Documentation	Inline comments explain non-obvious logic; README or header comments describe setup, dependencies, and usage. - a comment exists, no non-obvious logic to explain - no header comment - no readme	2	10%
Security Considerations	No hardcoded secrets; use of environment variables; input sanitization; helmet or CORS configured if applicable. - hardcoded secret in public view - no .env variables - no helmet/cors - no input sanitization	0	10%
Testing & Reliability	At least basic test cases provided (e.g., using Jest or Supertest); tests cover primary routes and edge cases. - no tests were implemented	0	10%

Total Weighted Score:	___**2.8**____ / 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.

#	File / Line	Severity	Category	Description / Observation
1	solution.js line 16	[✓ High] / Med / Low	Security	Hardcoded secret in public view / version control `if (password === "ILoveProgramming") {`
2	solution.js line 14, 23, 27	High / Med / [✓ Low]	Nitpick	modern approach uses async setup `app.get("/", (req, res) => {`

3	solution.js
	line 24, 29, 31
		High / Med / [✓ Low]	Nitpick	possibly better to use path.join
				`res.sendFile(__dirname +`
				`"/public/index.html");`

4	solution.js
	line 3, 4, 5
		High / Med / [✓ Low]	Redundancy	Doesn't __dirname already exist?
				`const __dirname =`
				`dirname(fileURLToPath(import.meta.url));`

5		High / Med / Low		
6	solution.js	[✓ High] / Med / Low	Security	No .env
7	solution.js	[✓ High] / Med / Low	Security	No helmet/cors implementation to prevent XSS / common attacks
8	solution.js function passwordCheck()	[✓ High] / Med / Low	Security	No input validation

6. EXPRESS.JS & JAVASCRIPT CHECKLIST

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

Category	Checklist Item	Y / N / N/A
Server Setup	Server listens on a configurable port (e.g., process.env.PORT)	Y
Server Setup	Entry point file is clearly identified (e.g., app.js or server.js)	N
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	Y
Middleware	Body-parser or express.json() configured for request parsing	Y
Middleware	Custom middleware is reusable (Y) and well-documented (N)	Y/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	N?
Dependencies	package.json lists all dependencies; no unused packages	Y
Dependencies	node_modules excluded via .gitignore	N

Category	Checklist Item	Y / N / N/A
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?

: simple and clean

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

: documentation, security implementations, best practices

<u>## Extra considerations for functionality/security</u>

: - currently no protection towards brute force attacks

- no session management:

- userIsAuthorized is a globally shared server variable

- all 'users' receive Auth=True once one user submits

correctly

– i.e. one browser submits correct password, sets Auth = True,

and receives secrets.html

– another browser submits ‘h’, Auth is already True, so receives

secrets.html even though erroneous

Suggested Learning Resources

Security Best Practices

- Use Helmet to secure your Express apps by setting various HTTP headers
- Use environment variables for configuration
- Implement proper error handling
- Use HTTPS in production
- Validate user input to prevent injection attacks
- Set appropriate CORS policies

from https://www.w3schools.com/nodejs/nodejs_express.asp

https://www.npmjs.com/package/express-session

https://www.geeksforgeeks.org/system-design/rate-limiting-in-system-design/

8. OVERALL ASSESSMENT

Grade	Range	Description
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.

+30% for it working and doing the intended task

Final Grade Assigned:	C+	**Numeric Score:**	__79_____ / 100

9. REQUIRED REVISIONS & ACTION ITEMS

List any mandatory changes the student must complete before resubmission.

#	Action Item	Priority	Due Date
1		High / Med / Low	
2		High / Med / Low	
3		High / Med / Low	
4		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:	Kendall Beam	**Date:**	2/14/26
Student Signature:		**Date:**	
Instructor Signature:		**Date:**	