Pillars Checkpoint Rubric

Language / Library Fluency (16 / 48)	Needs Work (1)	Fair (2)	Good (3)	Excellent (4)	Your Score
Free of syntax errors and runtime errors generated by tests	Tests do not run due to syntax errors in code	Tests run, with some tests failing due to syntax errors in code	Tests run, with at most one syntax error.	No syntax errors. Any spec failures are due to AssertionErrors rather than syntax errors.	
Properly manages control flow (conditionals, iteration, etc)	Conditionals are not present or check for the wrong criteria. Loops terminate improperly. Multiple code blocks are executed unintentionally	Several problems with control flow. Conditionals may have off-by-one errors.	Control flow is properly managed with one or two exceptions	Control flow properly managed. All logical expressions are appropriate	
Properly composes asynchronous actions	Improper or no usage of async/await. Code has cases where execution is non-deterministic due to lack of async control flow (race conditions).	Uses async/await and promises, but does not fully resolve or await for some async activity. Expects the return value of an async function to be something other than a promise.	Mostly proper application of async/await and promises. Attempts to get async values in an awkward fashion (i.e. trying to store them in variables in an outer scope)	All async functions declared properly, and all promises are awaited. Promise chains link properly.	
Uses ES6 language features (let, const, arrow functions, etc)	Uses var frequently. Sticks to older code patterns, or improperly applies ES6 features	Inconsistently applies new features. Mixes var with let and const. Sometimes uses the function keyword for callbacks, other times uses arrows.	Mostly appropriate use of new language features. Uses let when const would be more appropriate	Consistent and appropriate use of const and let. Uses arrows for callbacks when appropriate.	

Economy and Performance (12 / 48)	Needs Work (1)	Fair (2)	Good (3)	Excellent (4)
Avoids repetition in favor of reusable variables and functions (DRY)	Code frequently repeats itself. Copy/pastes the same thing over and over again rather than using a function	Code is clean in general, with some repetition/redundancy	Little repetition. May waste memory by creating data in functions that could be stored in its own variable	Functions and variables are DRY and improve the readability of the code.
Chooses the appropriate data structures and methods from JS for the job	Uses an array to store key-value data rather than an object. Uses for loops to transform data when .map or .filter would be appropriate	Properly employs arrays and objects. Does not always use array methods when the opportunity presents itself	Properly employs arrays and objects. Demonstrates some use of Array.prototype methods when appropriate	Properly employs objects and arrays. Uses Array.prototype methods with great facility
Doesn't perform unnecessary operations or declare unused variables or functions	Code has many unused/forgotten variables. Redundant or unnecessary function calls are made.	Code contains some unused variables or unnecessary work	One or two variables appear to have been forgotten, but code otherwise properly utilizes all of the values it reserves	All variables and functions are used appropriately.

Readability and Maintainability (10 / 48)	Needs Work (1)	Fair (2)	Good (3)	Excellent (4)
Variable and function names are meaningful and self-documenting	Most variable names are too short/long, or do not properly describe the data they contain.	Some variable and function names are meaningful. Variables containing promises are misleadingly named	Functions and variables are named appropriately, with some exceptions	Function and variable names appropriately communicate their contents/purpose, and consistently enhance readability.
Formatting is consistent (including whitespace, indentation, semicolons, bracket placement, etc)	Inconsistent formatting between functions. Lines of code are indented improperly, making it hard to read. Sometimes uses semicolons, other times doesn't.	Some inconsistency in style and indentation. Code is still readable	Code is readable with occasional inconsistencies in style.	Consistent adherence to a style guide/linter, with little to no deviations.

Cleanliness (12 / 48)	Needs Work (1)	Fair (2)	Good (3)	Excellent (4)
Handles errors consistently and properly	Errors are seldom handled - little to no use of try/catch or .catch	Some errors are handled, while others are neglected. In Express, does not pass errors to next	Mostly handles errors with few exceptions. In Express, passes errors to next with few exceptions.	Uses try/catch and .catch consistently. In Express, errors are properly given to next.
Doesn't contain commented out code (except to show incomplete thoughts) or console.log statements	Code is difficult to read due to the presence of large blocks of commented-out code. Code contains many console.log statements or a debugger statement	Some console.logs are leftover. Some commented out code remains but it doesn't inhibit readability	Code is mostly clean, though some comments/console.logs may have been left over or forgotten	No commented out code. No console.logs remain in source code
Conforms to best practices and avoids use of deprecated features, anti-patterns or anything known to introduce bugs	Uses features like eval and the arguments object. Creates objects/arrays by saying new Object or new Array, rather than object/array literals. Depends on the global scope. Initializes state with props. Depends on hoisting to use variables before they're declared	Uses the arguments object. Depends on the global scope. Uses variables before they're declared	Code contains one or two bad practices, but is otherwise in good shape	Code is free of bad habits and adheres to best practices

Overall

	You Got	Out Of	Percentage	
Specs Passing: (78%)		29	0	
Extra Credit Passing: (up to 3% added)		1	0	(will be 0 or 3)
Rubric Total: (25%)	0	48	0	

Final Grade: ((specs% + EC%) * .78) + (rubric% * .25) 0

Comments