PHASE-BY-PHASE AUDIT CHECKLIST

PHASE 1: CORE TRANSPILER (JavaScript to Lua)

FUNCTIONAL REQUIREMENTS

- [ ] **String Concatenation**: `+` operator correctly transpiled to `..`
- [ ] **Logical Operators**: `||` and `&&` correctly transpiled to `or` and `and`
- [ ] **Equality Operators**: `===` and `!==` correctly transpiled to `==` and `~=`
- [ ] **Variable Declarations**: `let`, `const`, `var` correctly transpiled to `local`
- [ ] **Function Declarations**: JavaScript functions correctly transpiled to Lua functions
- [ ] **Control Flow**: `if`, `for`, `while` statements correctly transpiled
- [ ] **Runtime Library**: Console, JSON, Math objects properly integrated

TECHNICAL REQUIREMENTS

- [ ] **Parser Accuracy**: AST generation is correct for all supported syntax
- [ ] **Code Generation**: Generated Lua code is syntactically correct
- [ ] **Error Handling**: Clear error messages for unsupported syntax
- [ ] **Memory Management**: Parser doesn't leak memory during transpilation

TEST COVERAGE

- [ ] **Unit Tests**: All transpiler functions have unit tests
- [ ] **Integration Tests**: End-to-end transpilation tests
- [ ] **Edge Cases**: Malformed input, edge syntax cases
- [ ] **Performance Tests**: Large file transpilation benchmarks

PHASE 2: RUNTIME SYSTEM

FUNCTIONAL REQUIREMENTS

- [ ] **Console Object**: `log`, `error`, `warn` methods work correctly
- [ ] **JSON Object**: `parse` and `stringify` methods functional
- [ ] **Math Object**: All mathematical functions available
- [ ] **Memory Management**: Proper allocation and deallocation
- [ ] **Garbage Collection**: Memory cleanup works correctly
- [ ] **Error Propagation**: JavaScript-style error handling

TECHNICAL REQUIREMENTS

- [ ] **Memory Safety**: No buffer overflows or memory corruption
- [ ] **Performance**: Runtime overhead is minimal
- [ ] **Compatibility**: Works with LuaJIT and standard Lua
- [ ] **Resource Management**: Proper cleanup of resources

TEST COVERAGE

- [ ] **Runtime Tests**: All runtime functions tested
- [ ] **Memory Tests**: Memory leak detection and prevention

- [ ] **Stress Tests**: High-load scenarios
- [ ] **Compatibility Tests**: Multiple Lua versions

PHASE 3: ADVANCED FEATURES

FUNCTIONAL REQUIREMENTS

- [ ] **Metaclass System**: Object-oriented programming support
- [ ] **Pattern Matching**: Advanced pattern matching capabilities
- [ ] **Memory Management**: Advanced memory allocation strategies
- [ ] **Concurrency**: Multi-threading and async support
- [ ] **Error Handling**: Comprehensive error handling with stack traces

TECHNICAL REQUIREMENTS

- [ ] **OOP Implementation**: Classes, inheritance, polymorphism
- [ ] **Pattern Engine**: Efficient pattern matching algorithms
- [ ] **Thread Safety**: Concurrent access safety
- [ ] **Performance**: Advanced features don’t degrade performance

TEST COVERAGE

- [ ] **OOP Tests**: Class hierarchies, inheritance chains
- [ ] **Pattern Tests**: Complex pattern matching scenarios
- [ ] **Concurrency Tests**: Multi-threaded execution
- [ ] **Integration Tests**: Advanced features work together

PHASE 4: ECOSYSTEM INTEGRATION

FUNCTIONAL REQUIREMENTS

- [ ] **Package Manager**: Install, update, remove packages
- [ ] **IDE Integration**: Language server, syntax highlighting
- [ ] **Build System**: Compilation, optimization, packaging
- [ ] **Testing Framework**: Unit testing, mocking, assertions
- [ ] **Debugging Tools**: Breakpoints, profiling, inspection

TECHNICAL REQUIREMENTS

- [ ] **Package Resolution**: Dependency management
- [ ] **Language Server**: LSP protocol compliance
- [ ] **Build Performance**: Fast compilation and optimization
- [ ] **Debug Accuracy**: Accurate debugging information

TEST COVERAGE

- [ ] **Package Tests**: Package installation and management
- [ ] **IDE Tests**: Language server functionality
- [ ] **Build Tests**: Build system reliability
- [ ] **Debug Tests**: Debugging tool accuracy

CROSS-PHASE INTEGRATION

INTEGRATION POINTS

- [ ] **Phase 1 → Phase 2**: Transpiled code runs correctly in runtime
- [ ] **Phase 2 → Phase 3**: Runtime supports advanced features
- [ ] **Phase 3 → Phase 4**: Advanced features work with ecosystem tools
- [ ] **Phase 4 → Phase 1**: Ecosystem tools can process transpiled code

SYSTEM-LEVEL TESTS

- [ ] **End-to-End**: Complete workflow from JS to running Lua
- [ ] **Performance**: System performance under load
- [ ] **Reliability**: System stability over time
- [ ] **Scalability**: System handles large projects

QUALITY METRICS

CODE QUALITY

- [ ] **Static Analysis**: No critical warnings (current: 209 warnings)
- [ ] **Code Coverage**: Minimum 95% coverage
- [ ] **Complexity**: Cyclomatic complexity within limits
- [ ] **Documentation**: All public APIs documented

PERFORMANCE METRICS

- [ ] **Transpilation Speed**: < 100ms per 1000 lines
- [ ] **Runtime Overhead**: < 10% compared to native Lua
- [ ] **Memory Usage**: < 50MB for typical projects
- [ ] **Startup Time**: < 1 second for runtime initialization

RELIABILITY METRICS

- [ ] **Crash Rate**: Zero crashes in normal operation
- [ ] **Error Recovery**: Graceful handling of all error conditions
- [ ] **Memory Leaks**: Zero memory leaks detected
- [ ] **Resource Cleanup**: All resources properly released

ACCEPTANCE CRITERIA

PHASE 1 ACCEPTANCE

- All transpiler tests pass
- Generated Lua code is syntactically correct
- Performance benchmarks met
- No memory leaks in parser

PHASE 2 ACCEPTANCE

- All runtime functions work correctly
- Memory management is stable
- Performance overhead is acceptable

- Compatibility with Lua versions verified

PHASE 3 ACCEPTANCE

- All advanced features implemented
- OOP system is complete
- Concurrency is thread-safe
- Pattern matching is efficient

PHASE 4 ACCEPTANCE

- Package manager is functional
- IDE integration works
- Build system is reliable
- Debugging tools are accurate

OVERALL SYSTEM ACCEPTANCE

- All phases integrate seamlessly
- System performance meets requirements
- No critical issues remain
- Documentation is complete
- Boss approval obtained

AUDIT INSTRUCTION: Each checkbox must be verified by at least 2 team members and signed off by a senior auditor before marking as complete.