Green Man Tavern - Validation Quick Start Guide

Get Started with Claude Code Audits in 30 Minutes

🚀 What You're About to Do

You're setting up an **automated quality assurance system** using Claude Code to:

1. ✅ Catch bugs and inconsistencies early
2. ✅ Prevent architectural drift
3. ✅ Maintain security and data integrity
4. ✅ Track code quality trends
5. ✅ Make informed deployment decisions

No setup required — just follow the steps below.

📋 STEP 1: Create Project Standards Document (5 min)

Create this file in your project root:

File: `.claude/project_standards.md`

```markdown
```

# Green Man Tavern - Code Standards & Conventions

## Core Architecture
- **LiveView → Context → Database**: No direct queries from LiveViews
- **User Ownership**: All user-scoped queries filter by user_id
- **Personality-Driven**: Characters have distinct voices
- **Greyscale Only**: No colors except for accessibility (#000-#FFF range)

## Key Components
- **Seven Seekers**: Student, Grandmother, Farmer, Robot, Alchemist, Survivalist, Hobo
- **Systems Flow Diagram**: Database-backed node visualization
- **Quest System**: Character-driven progression
- **Achievement System**: Milestone tracking
- **MindsDB Integration**: AI agents for character intelligence

## Naming Rules
- Modules: GreenManTavern.Domain.Module
- Contexts: GreenManTavern.Domain (no :context suffix)
- LiveViews: [Name]Live
- Tables: snake_case plural (users, user_systems)
- Functions: snake_case

## Database Rules
- All tables: inserted_at, updated_at timestamps
- All foreign keys: ON DELETE strategy specified
- All FK columns: Have indexes
- User-scoped data: Filter by user_id in queries

## Testing Minimum
- Contexts: >80% test coverage
- LiveViews: Happy path + error cases
- Integration: Full user journey tested

## Security Rules
- Authenticate before showing protected pages
- Filter all user-scoped data by current_user_id
- Validate all server-side form input
- Sanitize user content before display
- Never show other users' data

## Session Management
- Sessions timeout: 24 hours
- CSRF protection: On all forms
- HTTPS: Enforced in production
- Password requirements: Min 12 chars, strong

```
## Character Agent Integration
- Agents access only current_user's data
- Responses cached for 5 minutes
- Fallback to rule-based if MindsDB slow
- All agent calls logged for debugging
```

📋 STEP 2: Create Baseline Audit Tracker (5 min)

Create this file in your project root:

File: `AUDIT_BASELINE.md`

```
markdown
```

```markdown
# Green Man Tavern - Audit Baseline
**Establish your code quality starting point**

## Initial Audit Date: [TODAY]
Auditor: Claude Code

## Metrics Snapshot
- Codebase size: [X] lines
- Test coverage: [X]%
- Number of modules: [X]
- Number of LiveViews: [X]
- Number of tables: [X]
- Total tests: [X]

## Critical Issues Found
- [List any CRITICAL issues discovered]

## High-Priority Issues (Fix in next 2 weeks)
- [List top 5-10 items]

## Medium-Priority Issues (Fix in next month)
- [List next tier items]

## Reference Documents
- see: `.claude/audit_results/` for detailed reports
- see: `.claude/project_standards.md` for conventions

## Trends to Monitor
- Test coverage: Trending up/down/stable?
- Code quality: Improving/stable/declining?
- Performance: Getting faster/slower?
- Security: Any new issues?

## Next Review: [Date 1 week from today]
```

📋 STEP 3: Create Implementation Log (5 min)

Create this file in your project root:

File: `IMPLEMENTATION_LOG.md`

```markdown
```

# Green Man Tavern - Implementation Progress Log

## COMPLETED FEATURES ✅
- [x] User authentication (passwords, sessions)
- [x] Character system (all 7 Seekers defined)
- [x] Database schema (13 tables)
- [x] HyperCard UI components
- [x] Banner navigation
- [x] Left/right window layout
- [x] User profile management
- [ ] MindsDB agent integration
- [ ] Quest system
- [ ] Achievement system
- [ ] Opportunity detection algorithm
- [ ] Full integration testing

## CURRENT FOCUS
Working on: [What are you working on THIS WEEK?]
Expected completion: [Date]

## KNOWN ISSUES & WORKAROUNDS
1. **Issue**: [Description]
   Status: In progress / Blocked by [X] / Workaround: [Y]

2. **Issue**: [Description]
   Status: [Status]

## TECHNICAL DEBT
- [ ] [Item 1]: Would take ~[time] to fix
- [ ] [Item 2]: Would take ~[time] to fix
- [ ] [Item 3]: Would take ~[time] to fix

## AUDIT RESULTS SUMMARY
Last audit: [Date]
Key findings: [Brief summary]
Actions taken: [What did you fix?]

## METRICS TREND
Date	Coverage	Tests	Modules	Issues
Today	[X]%	[X]	[X]	[X]
-1wk	[X]%	[X]	[X]	[X]
-2wk	[X]%	[X]	[X]	[X]

## NEXT MILESTONES
1. [Milestone 1]: Target date [Date]

```
2. [Milestone 2]: Target date [Date]
3. [Milestone 3]: Target date [Date]

## DEPLOYMENT READINESS
- [ ] All tests passing
- [ ] No critical security issues
- [ ] Performance acceptable
- [ ] Documentation complete
- [ ] Backup strategy in place
```

📋 STEP 4: Create Audit Results Folder (1 min)

```bash
mkdir -p .claude/audit_results
```

This is where Claude Code will save audit reports.

🚀 STEP 5: Run Your First Audit (10 min)

Open Claude Code and run:

Run a comprehensive structural audit of the Green Man Tavern Phoenix LiveView + MindsDB codebase. Generate a detailed report covering:

PART 1: CODEBASE OVERVIEW
- Directory structure and file organization
- All LiveView modules (count, purposes, patterns)
- All Context modules (count, what they manage)
- Identify dead code or orphaned files
- Module dependency graph (any circular dependencies?)

PART 2: DATABASE SCHEMA INTEGRITY
- List all tables, their purposes, and which Context manages them
- Verify all foreign keys have ON DELETE strategies
- Check for missing indexes on foreign keys
- Identify normalized vs denormalized tables
- Find any tables not used by any code

PART 3: SECURITY & DATA ACCESS
- How is current_user determined across the app?
- Are all user-scoped queries filtering by user_id?
- Can user A access user B's data? (if so: CRITICAL issue)
- Any raw SQL queries that could have injection risks?
- Character agent data access: can agents see other users' data?

PART 4: ARCHITECTURE CONSISTENCY
- Are all LiveViews following LiveView → Context → Database pattern?
- Do all Contexts return {:ok, data} / {:error, reason}?
- Are all HTTP handlers properly protected?
- Do all user-facing features reference current_user?

PART 5: THE SEVEN SEEKERS SYSTEM
- Are all seven characters fully implemented?
- Do all characters have personality data for agent prompts?
- Is trust system properly tracked?
- Are character pages consistently styled?

PART 6: SYSTEMS FLOW DIAGRAM (LIVING WEB)
- Is database schema properly storing nodes and connections?
- Are there tests verifying the diagram integrity?
- Is opportunity detection algorithm implemented?
- Are all system types properly categorized?

PART 7: TESTING COVERAGE
- Total test count and coverage percentage
- Which Contexts have no tests?
- Which LiveViews have no tests?

```
- Are there integration tests covering full user journeys?

PART 8: NAMING CONVENTIONS
- Are all modules named GreenManTavern.Domain.Module?
- Are all LiveViews named with Live suffix?
- Are all tables named snake_case plural?
- Are all functions snake_case?

PART 9: DOCUMENTATION
- Does each module have @moduledoc?
- Do public functions have @doc with examples?
- Is the architecture documented?
- Is there a README for developers?

OUTPUT: Create a structured report with:
- Status for each section (pass/warning/critical)
- Specific issues found with line numbers/file paths
- Top 10 priorities ranked by severity
- Recommendations for each issue

Save the full report to: .claude/audit_results/initial_audit.md
```

🎯 STEP 6: Process the Results (5 min)

After Claude Code runs:

1. **Read the output** (will be very detailed)
2. **Identify CRITICAL issues** (🔴 — fix immediately)
3. **Create tickets** for High-priority issues (🟡)
4. **Note Medium/Low priorities** (🟢 — nice to have)

Update your `AUDIT_BASELINE.md` with key findings.

📅 STEP 7: Schedule Ongoing Checks

Add these to your calendar:

Weekly (Every Monday at 9:00 AM)

```
"Run weekly health check in Claude Code"
- Takes 5 minutes
- Identify any new issues
```

Bi-Weekly (Every Thursday at 9:00 AM)

```
"Run architecture drift check in Claude Code"
- Takes 10 minutes
- Verify patterns still being followed
```

Monthly (1st of month at 9:00 AM)

```
"Run full integration test in Claude Code"
- Takes 15 minutes
- Test complete user journey
```

Quarterly (End of each quarter)

```
"Run deep audit in Claude Code"
- Takes 30 minutes
- Compare to baseline, identify trends
```

📊 Example: What a Report Looks Like

Claude Code will produce something like:

```
AUDIT REPORT: Green Man Tavern Initial Audit
Date: 2025-01-20
Status: 🟡 ATTENTION NEEDED

PART 1: CODEBASE OVERVIEW
✅ Directory structure well organized
✅ 12 LiveView modules identified
✅ 8 Context modules identified
⚠️ Found 2 orphaned files: old_character_page.ex, legacy_form.ex
⚠️ Found 1 circular dependency: SystemsContext → CharsContext → SystemsContext

PART 2: DATABASE SCHEMA
✅ 13 tables, all with timestamps
✅ All foreign keys have ON DELETE CASCADE
⚠️ Missing index on user_systems.user_id (queried frequently)
⚠️ user_connections.user_id index exists but missing on type+user_id composite

PART 3: SECURITY & DATA ACCESS
⚠️ UserSystems LiveView: Queries systems but doesn't verify user ownership
🔴 CRITICAL: Direct Ecto.Repo query in CharacterLive - bypasses Context
⚠️ MindsDB context builder: Could expose other user's systems in rare cases

... [continued detailed analysis] ...

TOP 10 PRIORITIES:
1. 🔴 CRITICAL: Fix CharacterLive direct database query (1 hour)
2. 🔴 CRITICAL: Add user_id filter to UserSystems query (30 min)
3. 🟡 HIGH: Remove circular dependency (2 hours)
4. 🟡 HIGH: Add composite index on user_connections (15 min)
5. 🟡 HIGH: Test MindsDB context for data leaks (1 hour)
6. 🟡 HIGH: Add @doc comments to 15 public functions (1.5 hours)
7. 🟠 MEDIUM: Delete 2 orphaned files (5 min)
8. 🟠 MEDIUM: Add test for user isolation (1 hour)
9. 🟠 MEDIUM: Document Seven Seekers system (30 min)
10. 🟢 LOW: Add Monica theme to README (15 min)

RECOMMENDATIONS:
- Fix the 3 critical/high items immediately
- Schedule 2-hour refactoring session this week
- Add security tests to CI pipeline
- Document architecture patterns
```

✅ Checklist: First Week Setup

- [ ] Created `.claude/project_standards.md`
- [ ] Created `AUDIT_BASELINE.md`
- [ ] Created `IMPLEMENTATION_LOG.md`
- [ ] Created `.claude/audit_results/` folder
- [ ] Ran initial audit with Claude Code
- [ ] Read and understood initial audit results
- [ ] Created tickets for CRITICAL/HIGH issues
- [ ] Added weekly/monthly reminders to calendar
- [ ] Bookmarked `.claude/audit_results/` for easy access
- [ ] Shared Claude Code audit approach with any team members

---

🎯 Quick Reference: What to Do When

When you write code:

```
"Does this follow .claude/project_standards.md?"
- Use LiveView → Context → Database pattern?
- Filter by current_user_id for user data?
- Have tests?
- Documented?
```

Before committing:

```
"Run quick code review in Claude Code"
```

Every Monday morning:

```
"Run weekly health check in Claude Code"
```

Before deploying to production:

```
"Run pre-deployment checklist in Claude Code"
```

When something feels messy:

```
"Run refactoring guidance in Claude Code"
```

When a user reports a bug:

```
"Run security audit in Claude Code" (to ensure it's not a data leak)
"Run custom module audit in Claude Code" (for the affected module)
```

🆘 Troubleshooting

Q: Claude Code says "Module not found" or "File not found"

A: Claude Code needs to be able to see your file structure. Make sure your repository root is clean (no node_modules, build artifacts) and all code is in lib/, test/, priv/ directories.

Q: Audit takes a long time

A: Normal for first audit (~2-5 min). Subsequent audits are faster. You can run audits during breaks.

Q: Claude Code output is overwhelming

A: Focus on the CRITICAL issues (🔴) first. High-priority (🟡) in the next sprint. Medium/Low (🟢) can be batched for tech debt days.

Q: How do I know if I should deploy?

A: Check the Pre-Deployment Checklist report. If it says "🔴 DO NOT DEPLOY" — don't. If "✅ SAFE TO DEPLOY" or "⚠️ WITH CAUTION" — proceed as documented.

Q: What if I disagree with a recommendation?

A: Document it in `IMPLEMENTATION_LOG.md`. Keep notes on decisions made. Share with team if applicable.

Q: Can I run audits on just one module?

A: Yes! Use the "CUSTOM: Check Specific Module" prompt in the Claude Code prompts document. Specify the module name.

📈 Tracking Progress Over Time

Week 1 Baseline

```
Coverage: 45%
Tests: 120
Critical Issues: 3
Time to Deploy: 4+ hours
```

Week 4 (After fixes)

```
Coverage: 62%
Tests: 187
Critical Issues: 0
Time to Deploy: 1 hour
```

Week 12 (One quarter)

```
Coverage: 78%
Tests: 287
Critical Issues: 0
Time to Deploy: 30 min
```

Keep this trend visible to celebrate progress!

🎓 Learning Path

If you're new to this approach:

1. Read the "Quick Start Guide" (this document)
2. Complete all steps in "First Week Setup"
3. Run the initial audit
4. Fix the CRITICAL issues
5. Run weekly health checks
6. After 4 weeks, run a trend analysis

If you're integrating with a team:

1. Share `.claude/project_standards.md` with team
2. Explain why these standards matter
3. Show example audit report
4. Agree on what "Critical" vs "High" means for your team
5. Add audit results to your team's documentation

If you want to go deeper:

1. Read "Structural Validation & Consistency Strategy" (full document)
2. Understand each audit type's purpose

3. Customize prompts for your team's needs
4. Build additional checks as needed

🚀 You're Ready!

You now have:

✅ **A quality assurance system** powered by Claude Code
✅ **Documented standards** all code should follow
✅ **Baseline metrics** to track progress
✅ **Regular checkpoints** to catch issues early
✅ **Clear deployment criteria** before shipping
✅ **A way to prevent technical debt** from accumulating

Next Step:

Open Claude Code and run your first audit right now.

It will take 10-15 minutes, and you'll immediately know where your project stands.

📚 Reference: All Documents

You now have these documents:

1. **Structural Validation & Consistency Strategy** (Full Strategy)
   - Comprehensive overview of the entire approach
   - Part 1: Initial deep audits (6 types)
   - Part 2: Ongoing periodic checks (weekly/bi-weekly/monthly/quarterly)
   - Part 3: Setup & maintenance documentation
   - Part 4: Claude Code workflow
   - Part 5: Documentation for each check
2. **Claude Code Prompts** (Ready-to-Use)
   - 13 different prompts
   - Copy/paste directly into Claude Code
   - Each prompt is self-contained and specific
3. **Quick Start Guide** (This Document)
   - Get up and running in 30 minutes

- Step-by-step setup
- Examples and troubleshooting
- Learning paths

Save These Locally:

```bash
# Copy to your project
cp "Structural Validation & Consistency Strategy.md" docs/VALIDATION_STRATEGY.md
cp "Claude Code Prompts (Ready-to-Use).md" .claude/AUDIT_PROMPTS.md
cp "Quick Start Guide.md" docs/QUICK_START.md
```

💾 File Structure After Setup

```
green-man-tavern/
├── .claude/
│   ├── project_standards.md       ← Created in Step 1
│   ├── AUDIT_PROMPTS.md           ← Reference all prompts
│   └── audit_results/             ← Created in Step 4
│       ├── initial_audit.md
│       ├── weekly_2025_01_20.md
│       ├── monthly_2025_02_01.md
│       └── quarterly_2025_03_31.md
│
├── docs/
│   ├── VALIDATION_STRATEGY.md     ← Full strategy document
│   └── QUICK_START.md             ← This guide
│
├── AUDIT_BASELINE.md              ← Created in Step 2
├── IMPLEMENTATION_LOG.md          ← Created in Step 3
├── mix.exs
├── lib/
│   └── green_man_tavern/
├── test/
└── priv/
```

🎯 Success Metrics

Your validation system is **working well** when:

✅ **Weekly audits** run quickly (5 min) with no surprises
✅ **No new Critical issues** appear unexpectedly
✅ **Test coverage** trending upward over time
✅ **Architecture patterns** consistent across all code
✅ **Security audits** finding zero authentication/authorization issues
✅ **Pre-deployment checklist** green before shipping
✅ **Developers** confident in code quality
✅ **Refactoring decisions** data-driven (based on audit results)

📞 Support

If you get stuck:

1. **Check the troubleshooting section** (earlier in this doc)
2. **Review the full strategy document** for more context
3. **Re-read the specific audit prompt** to understand what Claude Code is checking
4. **Add your own notes** to `IMPLEMENTATION_LOG.md` for future reference

🎉 Congratulations!

You've set up a **professional-grade code quality assurance system** for Green Man Tavern.

You're now equipped to:

- Catch bugs before they reach users
- Prevent architectural drift
- Make data-driven deployment decisions
- Maintain consistent code quality
- Track progress over time
- Onboard new developers confidently

Your next action:

Run your first audit in Claude Code RIGHT NOW. Don't wait.

Takes 15 minutes. You'll immediately understand your codebase's health.

Version: 1.0

Date: Today

Status: Ready to implement

Questions? Check `.claude/project_standards.md` or the full strategy document.