Green Man Tavern - Structural Validation & Consistency Strategy

Using Claude Code for Codebase Health Checks

🎯 Overview

This document outlines a comprehensive strategy to use **Claude Code** as your quality assurance tool, ensuring the Green Man Tavern project maintains:

- **Architectural consistency** across all modules
- **Database integrity** (schema, relationships, migrations)
- **Security posture** (authentication, data access, injection vulnerabilities)
- **Performance standards** (N+1 queries, indexing, caching)
- **Code quality** (naming, documentation, duplication)
- **Integration correctness** (all modules working together seamlessly)

📋 Part 1: Initial Deep Audit (Run Once)

Objective

Establish a baseline understanding of the current codebase health and identify any existing issues before further development.

Phase 1a: Codebase Overview Audit

Claude Code Prompt:

```
Analyze the entire Green Man Tavern Phoenix LiveView + MindsDB project codebase.
Generate a comprehensive overview including:

1. **File Structure Analysis**:
   - Describe directory organization
   - Identify any orphaned files or dead code
   - List all LiveView modules and their relationships
   - Map all Contexts (business logic layers)

2. **Module Dependencies**:
   - Create a dependency graph showing which modules import which
   - Identify circular dependencies
   - Flag any modules that are too tightly coupled
   - Suggest candidates for extraction/refactoring

3. **Database Schema Audit**:
   - List all tables and relationships
   - Check for orphaned tables (not used by any Context)
   - Verify all foreign keys have ON DELETE strategies
   - Identify missing indexes on frequently queried fields
   - Check for normalization issues
   - Verify timestamp columns are present on all tables

4. **LiveView Architecture**:
   - How many LiveView modules exist?
   - Do they follow consistent patterns (mount, handle_event, handle_info)?
   - Are they properly organized (one concern per module)?
   - Any LiveViews doing business logic (should be in Contexts)?

5. **Naming Convention Consistency**:
   - Module names: Do they follow `GreenManTavern.Domain.Module` pattern?
   - Context names: All ending in context function names?
   - Database table names: All snake_case plural?
   - LiveView names: All ending with `Live`?
   - Test files: All following `*_test.exs` pattern?

Output format: JSON-like checklist with pass/fail/warning status
```

Phase 1b: Security & Data Access Audit

Claude Code Prompt:

```
Security audit for Green Man Tavern Phoenix + MindsDB codebase.

Focus on:

1. **Authentication & Authorization**:
   - How is current_user determined?
   - Are all LiveViews checking user ownership before showing data?
   - Can user A access user B's systems/quests/achievements?
   - Are there any unprotected endpoints?

2. **Data Access Patterns**:
   - Do LiveViews query database directly, or through Contexts?
   - Any raw SQL queries that could be SQL injection vectors?
   - How is user-uploaded data (if any) sanitized?
   - Are all list/show/edit/delete operations checking user ownership?

3. **MindsDB Agent Context**:
   - How is user context passed to MindsDB agents?
   - Can an agent see other users' data?
   - Are agent responses sanitized before displaying to user?
   - What happens if an agent hallucinate or returns malicious content?

4. **Session Management**:
   - How long do sessions last?
   - Are there CSRF protections on all forms?
   - Is HTTPS enforced in production config?

5. **Specific Concerns**:
   - User profiles: Can users see/edit others' profiles?
   - Systems diagram: Is it truly user-specific in database queries?
   - Achievements: Are they isolated per user?
   - Character agents: Do they only access relevant user's data?

Output: Risk assessment with severity (Critical/High/Medium/Low) and remediation steps
```

Phase 1c: Architecture Consistency Audit

Claude Code Prompt:

Architecture consistency audit for Green Man Tavern.

Verify these key architectural decisions are consistently applied:

1. **Seven Seekers Character System**:
   - How many character modules/tables exist?
   - Are all seven characters (Student, Grandmother, Farmer, Robot, Alchemist, Survivalist, Hobo) fully implemented?
   - Do all character interactions follow the same pattern?
   - Are personality prompts consistent and accessible to MindsDB agents?

2. **Systems Flow Diagram (Living Web)**:
   - Is the diagram data model consistent with database schema?
   - Are nodes properly typed (resource/process/storage)?
   - Are connections properly tracked (active/potential)?
   - Is the opportunity detection algorithm implemented or planned?

3. **Quest System**:
   - Are all quest types implemented (tutorial, implementation, maintenance, learning, community, challenge)?
   - Do quests properly reference systems and achievements?
   - Is quest difficulty properly calculated?
   - Are quest rewards (XP, achievements) working correctly?

4. **User Progression**:
   - Is XP calculation consistent?
   - Do level thresholds match design spec?
   - Are achievements properly gating features?
   - Is trust system for characters functional?

5. **HyperCard Aesthetic**:
   - Are all UI components using consistent greyscale palette?
   - Are windows styled consistently?
   - Is typography consistent (Monaco, pixel-perfect)?
   - Are there any color violations (should be greyscale only)?

6. **MindsDB Integration**:
   - Are all agents properly configured?
   - Do agents access the same knowledge base?
   - Are context injection patterns consistent?
   - Is response caching implemented?

Output: Gap analysis showing what's implemented vs. what's missing

Phase 1d: Performance & Database Efficiency Audit

Claude Code Prompt:

```
Performance and database efficiency audit for Green Man Tavern.

Analyze:

1. **Query Efficiency**:
   - Which LiveViews query the database?
   - Any obvious N+1 query problems? (e.g., loop over systems then query connections for each)
   - Are associations being preloaded with `preload` or `:include`?
   - Are there any queries that could be batched?
   - Which queries run on every page load (candidates for caching)?

2. **Database Indexing**:
   - Do all foreign key columns have indexes?
   - Are there indexes on frequently filtered/sorted columns?
   - Are there composite indexes where appropriate?
   - Are there unused indexes that could be removed?

3. **LiveView Socket Assigns**:
   - How much data is stored in socket.assigns?
   - Are there large collections being stored unnecessarily?
   - Should any of this be moved to temporary_assigns?
   - Are streams being used for paginated data?

4. **Caching Opportunities**:
   - MindsDB agent responses: Are they cached?
   - Character profiles: Cached?
   - Systems library: Cached?
   - Opportunity detection: Cached?
   - What TTL should each have?

5. **Real-time Features**:
   - Which features use PubSub?
   - Could any PubSub messages be batched?
   - Are subscription/unsubscribe calls balanced?

Output: Performance recommendations ranked by impact (high/medium/low)
```

Phase 1e: Testing Coverage Audit

Claude Code Prompt:

```
Test coverage and quality audit for Green Man Tavern.

Analyze:

1. **Test Existence**:
   - How many ExUnit tests exist?
   - What's the coverage percentage?
   - Are all Contexts tested?
   - Are all LiveViews tested?
   - Are helpers/utilities tested?

2. **Test Quality**:
   - Do tests verify behavior, not implementation?
   - Are fixtures used consistently?
   - Are there test factories for complex objects?
   - Do integration tests cover the happy path AND error cases?

3. **Missing Tests**:
   - Which Contexts have no tests?
   - Which LiveViews have no tests?
   - What about authentication tests?
   - What about authorization tests?
   - Are MindsDB agent integrations tested?

4. **Test Patterns**:
   - Are tests organized by module?
   - Do they follow consistent naming conventions?
   - Are there helper functions to reduce duplication?
   - Do tests create appropriate test data?

5. **CI/CD Readiness**:
   - Is there a CI pipeline (GitHub Actions, CircleCI)?
   - Do all tests pass locally?
   - Are there any flaky tests?
   - What's the test execution time?

Output: Test coverage report with recommendations for priority areas
```

Phase 1f: Documentation Audit

Claude Code Prompt:

```
Documentation and onboarding audit for Green Man Tavern.

Check:

1. **Module Documentation**:
   - Do all modules have @moduledoc comments?
   - Do all public functions have @doc comments with examples?
   - Are complex algorithms explained?
   - Are edge cases documented?

2. **Architecture Documentation**:
   - Is there a README explaining the project?
   - Is there an architecture guide explaining components?
   - Are integration points documented?
   - Is the database schema documented?
   - Are the Seven Seekers' personalities documented?

3. **Developer Onboarding**:
   - Could a new developer understand the codebase?
   - Are local setup instructions clear?
   - Are there examples of common tasks?
   - Is there a glossary of domain terms?

4. **API Documentation**:
   - Are Context functions documented?
   - Are LiveView events documented?
   - Are data structures documented?

Output: Documentation gaps ranked by importance
```

📋 Part 2: Ongoing Periodic Checks

Weekly Structural Health Check

Run every Monday morning

Claude Code Prompt:

```
Weekly structural health check for Green Man Tavern.

Compare current codebase to last week's baseline and report:

1. **New modules added**: Verify they follow naming conventions and patterns
2. **Modified migrations**: Ensure all changes have down/rollback steps
3. **Test coverage**: Has it improved or regressed?
4. **Code duplication**: Any new duplicated patterns?
5. **Dependencies**: Any new external dependencies added? Are they justified?
6. **Performance regressions**: Any obvious new N+1 queries?
7. **Security concerns**: Any new user input handling that needs sanitization?

Output: Checklist format with green/yellow/red indicators
```

Bi-Weekly Architecture Drift Check

Run every other Thursday

Claude Code Prompt:

```
Architecture drift detection for Green Man Tavern.

Verify these specific architectural contracts are still being followed:

1. **LiveView → Context → Database pattern**:
   - Are any LiveViews doing database queries directly?
   - Are any Contexts doing business logic that should be in LiveViews?

2. **Consistent error handling**:
   - Are all database operations wrapped in {:ok, data} / {:error, reason}?
   - Are all LiveView events handling errors consistently?
   - Is user feedback consistent?

3. **HyperCard aesthetic**:
   - Scan all CSS files: Any non-greyscale colors?
   - Check all components: Are titles using Monaco font?
   - Verify window styling: All using consistent Mac chrome?

4. **Character system**:
   - Are all seven characters equally implemented?
   - Are personality prompts being used in agent calls?
   - Is trust system being incremented properly?

5. **Systems diagram consistency**:
   - Are all system types properly categorized?
   - Are connections properly validated before save?
   - Is opportunity detection algorithm producing sensible results?

Output: Drift report with before/after comparisons
```

Monthly Integration Test

Run on the 1st of each month

Claude Code Prompt:

Full integration test for Green Man Tavern - all modules together.

Simulate a complete user journey and verify:

1. **User Registration & Onboarding**:
   - Create test user account
   - Navigate robot onboarding (profile questions)
   - Verify profile data stored correctly
   - Verify user can select primary character

2. **Systems Flow Diagram**:
   - Add a system to their diagram
   - Create a connection between systems
   - Verify opportunity detection suggests improvements
   - Modify existing connection
   - Delete a system

3. **Character Interaction**:
   - Chat with primary character (mock MindsDB)
   - Verify character personality in response
   - Verify user context passed to agent
   - Verify responses stored in conversation history

4. **Quest System**:
   - View available quests
   - Accept a quest
   - Mark quest complete
   - Verify XP awarded
   - Check achievement unlocked (if applicable)

5. **Progression**:
   - Verify character trust level increased
   - Verify new level/XP displayed
   - Verify unlocked features now available

6. **Data Persistence**:
   - Log out and log back in
   - Verify all data still there
   - Verify systems diagram unchanged
   - Verify progress not reset

Output: Pass/fail for each step with detailed logs

Quarterly Deep Audit

Run at end of each quarter

Claude Code Prompt:

```
Quarterly deep audit for Green Man Tavern.

Comprehensive re-evaluation of:

1. **Database Health**:
   - Any orphaned data?
   - Migration consistency?
   - Backup and recovery process working?
   - Data integrity constraints being enforced?

2. **Code Quality Trend**:
   - Is code getting better or worse?
   - Which modules have highest complexity?
   - Which modules change most frequently?
   - Which modules have zero tests?

3. **Technical Debt**:
   - What's accumulating?
   - What should be refactored before it becomes urgent?
   - Any deprecated patterns still being used?
   - Any abandoned features cluttering the codebase?

4. **Performance Trends**:
   - Average page load time?
   - Average MindsDB agent response time?
   - Database query performance?
   - Any memory leaks?

5. **Security Posture**:
   - Any new vulnerability types introduced?
   - Is authentication still solid?
   - Are permissions properly enforced?
   - Any new attack surfaces?

Output: Executive summary with priorities for next quarter
```

📋 Part 3: Setup & Maintenance Documentation

Required Setup Files

Create these files in your project root:

`.claude/project_standards.md` (Reference documentation)

```markdown
```

```
# Green Man Tavern - Code Standards & Conventions

## Architecture Principles
- **LiveView → Context → Database**: LiveViews never query database directly
- **User Ownership**: All queries filter by current_user_id
- **Personality-Driven**: Characters have distinct voices reflected in agent prompts
- **Visual Consistency**: Greyscale only (#000, #333, #666, #999, #CCC, #EEE, #FFF)

## Naming Conventions
- Modules: GreenManTavern.Domain.Module
- Contexts: GreenManTavern.Domain (e.g., GreenManTavern.Characters)
- LiveViews: Module ending with Live
- Tables: snake_case, plural (users, user_systems, user_connections)
- Functions: snake_case, verb+noun pattern

## Database Conventions
- All tables have timestamps: inserted_at, updated_at
- All foreign keys have ON DELETE strategy
- All foreign key columns are indexed
- User-scoped tables must filter by user_id

## Seven Seekers Characters
- The Student (Knowledge Seeker)
- The Grandmother (Elder Wisdom)
- The Farmer (Food Producer)
- The Robot (Tech Integration)
- The Alchemist (Plant Processor)
- The Survivalist (Resilience Expert)
- The Hobo (Nomadic Wisdom)

## Testing Requirements
- All Contexts: >80% coverage
- All LiveViews: happy path + error cases
- Integration tests: full user journeys
- MindsDB agent calls: mocked in tests

## Security Requirements
- All user-scoped data filtered by current_user_id
- No user-uploaded files without sanitization
- All form inputs validated server-side
- HTTPS enforced in production
- Sessions timeout after 24 hours
```

`AUDIT_BASELINE.md` (Save results from initial audits)

```
markdown
```

```markdown
# Green Man Tavern - Audit Baseline
Date: [TODAY]
Auditor: Claude Code

## Initial Metrics
- Total modules: [X]
- Total lines of code: [X]
- Test coverage: [X]%
- Database tables: [X]
- LiveView modules: [X]
- Context modules: [X]

## Critical Issues Found
- [List any critical security/architecture issues]

## High-Priority Recommendations
- [List top 5-10 items]

## Reference Audits
- See audit results in `.claude/audits/` folder

## Next Review: [Date]
```

IMPLEMENTATION_LOG.md (Track what you've built)

```
markdown
```

```
# Implementation Log

## Completed Features
- [x] User authentication
- [x] Character system (7 Seekers defined)
- [x] Systems flow diagram database
- [x] HyperCard UI components
- [x] Database module for user profiles
- [ ] MindsDB agent integration
- [ ] Quest system implementation
- [ ] Achievement system implementation
- [ ] Opportunity detection algorithm

## Known Issues
- [Issue 1]: [Status/workaround]
- [Issue 2]: [Status/workaround]

## Next Milestones
1. [By date]: Complete MindsDB agents
2. [By date]: Implement quest system
3. [By date]: Launch soft beta
```

📋 Part 4: Claude Code Workflow

Daily Workflow

Morning (5 min):

```
claude-code "Quick health check: Any new files that need review?
Any obvious code quality issues in files changed yesterday?"
```

After coding session (10 min):

```
claude-code "Review my changes today. Do they follow the conventions
in .claude/project_standards.md? Are there any new security concerns?"
```

Before committing:

```
claude-code "Final review: Do these changes maintain architectural
consistency? Any edge cases I might have missed?"
```

Weekly Workflow

Monday morning:

- Run Phase 1b's "Weekly Structural Health Check"
- Update AUDIT_BASELINE.md with new metrics
- Review test coverage report

Friday afternoon:

- Run any failing tests through Claude Code for diagnosis
- Ask: "What should I focus on improving next week?"

Milestone Workflow

Before major feature launch:

- Run Phase 1c's "Architecture Consistency Audit"
- Run Phase 1d's "Performance & Efficiency Audit"
- Run Phase 1e's "Testing Coverage Audit"
- Fix critical issues
- Document changes in IMPLEMENTATION_LOG.md

Before user launch:

- Run Phase 1b's "Security & Data Access Audit"
- Run Phase 2's "Monthly Integration Test"
- Get final security clearance
- Document any known limitations

📋 Part 5: Documentation for Each Check

What Claude Code Outputs

Each check produces:

1. **Executive Summary** - 1-2 paragraphs of key findings
2. **Detailed Report** - Item-by-item breakdown
3. **Severity Ratings** - Critical/High/Medium/Low

4. **Actionable Recommendations** - Ranked by impact
5. **Code Examples** - Shows patterns to follow/avoid
6. **Specific File References** - Line numbers and modules

How to Process Results

1. **Read executive summary** - Understand overall status
2. **Prioritize Critical items** - Address before other work
3. **Create tickets** - For High/Medium items
4. **Update project_standards.md** - If new patterns discovered
5. **Commit changes** - Document audit findings in commit message
6. **Archive report** - Save to `.claude/audit_results/[date].md`

🎯 Success Criteria

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

✅ **Weekly checks** consistently pass with no new Critical issues
✅ **Architecture consistency** maintained across all modules
✅ **Test coverage** stays above 75%
✅ **Security audits** find zero authentication/authorization breaches
✅ **Integration tests** all pass before each deployment
✅ **Performance** stays within acceptable baselines (< 2s page load)
✅ **Code quality** improves or stays stable quarter-over-quarter
✅ **New developers** can onboard using documentation

📋 Quick Reference: Audit Checklist

Print this and post it somewhere:

```
INITIAL AUDIT (Run Once - Do Today)
☐ Phase 1a: Codebase Overview
☐ Phase 1b: Security & Data Access
☐ Phase 1c: Architecture Consistency
☐ Phase 1d: Performance & Efficiency
☐ Phase 1e: Testing Coverage
☐ Phase 1f: Documentation

Setup:
☐ Create .claude/project_standards.md
☐ Create AUDIT_BASELINE.md with results
☐ Create IMPLEMENTATION_LOG.md
☐ Create .claude/audit_results/ folder

ONGOING CHECKS
☐ Weekly (Monday morning): Structural Health
☐ Bi-weekly (Thursday): Architecture Drift
☐ Monthly (1st): Integration Test
☐ Quarterly: Deep Audit

COMMITTED
☐ Project standards documented
☐ Baseline established
☐ Review process defined
☐ Team understands gates before launch
```

🚀 Next Steps

1. **This week**: Run Phase 1a-f audits to establish baseline
2. **Create the three reference files** (.claude/project_standards.md, AUDIT_BASELINE.md, IMPLEMENTATION_LOG.md)
3. **Set calendar reminders** for weekly, bi-weekly, monthly, and quarterly checks
4. **Review first results** with Claude Code to understand what needs priority
5. **Begin addressing Critical issues** before any feature development

---

Version: 1.0
Last Updated: Today
Next Review: [After completing Phase 1 audits]