Green Man Tavern - Claude Code Prompts

Ready-to-Copy Prompts for Structural Validation

🚀 Quick Start: Run This First (Today)

MASTER INITIAL AUDIT PROMPT

Copy this entire prompt into Claude Code:

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?
- User-uploaded content handling (if applicable)
- 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?
- Are MindsDB agent calls properly mocked in tests?

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?
- Are domain terms glossed?

OUTPUT FORMAT:
- Create a structured JSON report
- For each section: {status: "ok" | "warning" | "critical", details: [...], recommendations: [...]}
- At the end: Ranked list of top 10 priorities (Critical first)
- Suggest which issues block further development vs. which are nice-to-have

SAVE THIS REPORT TO: .claude/initial_audit_[DATE].md

📆 WEEKLY: Structural Health Check

Run every Monday morning

Weekly health check for Green Man Tavern - quick status update.

Check these specific items and report status (pass/warning/fail):

1. NEW FILES:
   - Any new modules added this week?
   - Do they follow naming conventions?
   - Are they properly integrated?

2. TEST COVERAGE:
   - Has it improved from last week?
   - Any new test failures?

3. OBVIOUS ISSUES:
   - Any TODO/FIXME comments added?
   - Any obvious code smells in new code?

4. SECURITY:
   - Any new user input handling without sanitization?
   - Any new queries that might leak user data?

5. PERFORMANCE:
   - Any obvious N+1 query patterns in new code?
   - Any new database calls in loops?

OUTPUT: Simple checklist format with emojis
- ✅ All good
- ⚠️ Minor issue
- 🔴 Critical issue

End with: "Next priority: [X]"

📅 BI-WEEKLY: Architecture Drift Detection

Run every other Thursday

Architecture drift detection for Green Man Tavern.

Verify these specific patterns are STILL being followed throughout
the codebase:

CONTRACT 1: LiveView → Context → Database
- Scan all LiveView files for direct Ecto.Repo calls
- If found: List them and flag as violation
- Check: Do LiveViews always call Context functions?

CONTRACT 2: User Ownership on All User-Scoped Data
- List all queries on users_systems, user_connections, user_profiles
- Verify every single query filters by current_user_id
- Find any that don't and list them as vulnerabilities

CONTRACT 3: Consistent Error Handling
- Do all database operations return {:ok, data} / {:error, reason}?
- Do all LiveView events handle errors with flash messages?
- Find exceptions to this pattern

CONTRACT 4: Seven Seekers Consistency
- Are all 7 characters equally implemented?
- Do they all have personality prompts?
- Is trust tracking consistent?
- Are they all accessible from the UI?

CONTRACT 5: HyperCard Aesthetic
- Scan CSS/styling: Any non-greyscale colors?
- Check typography: Are we using consistent fonts?
- Window styling: All using same chrome pattern?
- Find any violations

CONTRACT 6: Systems Diagram
- Can users add systems?
- Can users create connections?
- Are opportunities being detected?
- Is the diagram data consistent with database?

OUTPUT FORMAT:
Section by section: [Contract Name]
- Status: ✅ Maintained / ⚠️ Drifting / 🔴 Broken
- Details: List any violations found
- Recommendation: What to fix

Include: "Would you recommend refactoring any of these contracts?"

📅 MONTHLY (1st): Full Integration Test

Run on the 1st of every month

Full end-to-end integration test for Green Man Tavern.

Simulate a complete user journey step by step. For each step,
output whether it would pass or fail and why:

JOURNEY:

Step 1: New User Registration
- User registers account with email/password
- VERIFY: User created in database
- VERIFY: Session created
- VERIFY: Redirected to onboarding

Step 2: Robot Onboarding
- Robot asks profile questions
- User provides location, space type, skill level
- VERIFY: Responses saved to user_profile
- VERIFY: System suggestions generated appropriately

Step 3: Character Selection
- User chooses primary character (e.g., The Grandmother)
- VERIFY: Character set as primary_character_id
- VERIFY: Character trust_level initialized

Step 4: Systems Diagram
- User adds first system (e.g., herb garden)
- VERIFY: System created in user_systems
- VERIFY: Node appears in diagram
- User adds second system (e.g., kitchen)
- VERIFY: System created
- User creates connection: herb garden → kitchen
- VERIFY: Connection saved, shows as "active"
- User receives opportunity to add drying system
- VERIFY: Opportunity detection working

Step 5: Character Chat
- User selects The Grandmother
- User types: "How do I preserve my herbs?"
- VERIFY: Message stored in conversation_history
- VERIFY: Request sent to MindsDB agent
- VERIFY: Agent response received and displayed
- VERIFY: Character personality evident in response
- VERIFY: User context was injected (knows about herb garden)

Step 6: Quest System
- User accepts a quest

- VERIFY: Quest status changed to "active"
- User completes quest
- VERIFY: Status changed to "completed"
- VERIFY: XP awarded
- VERIFY: If achievement unlocked: shown to user

Step 7: Progression
- VERIFY: Character trust level increased
- VERIFY: User XP total increased
- VERIFY: If leveled up: shown to user
- VERIFY: If achievement earned: displayed

Step 8: Data Persistence
- User logs out
- VERIFY: Session cleared
- User logs back in
- VERIFY: All data intact (systems, quests, progress)
- VERIFY: Systems diagram unchanged
- VERIFY: Conversation history preserved

Step 9: Database Module
- User selects "Database" from banner
- VERIFY: Right window shows user's personal data
- User edits profile (change location)
- VERIFY: Changes saved
- User views systems list
- VERIFY: All systems listed
- VERIFY: Can edit or delete systems

RESULTS:
- Pass/Fail for each step
- If any step fails: detailed diagnosis
- If all pass: "Ready for deployment"
- If some fail: Prioritized list of fixes

END WITH: "Deployment blockers: [list any]"

📅 QUARTERLY: Deep Audit

Run at end of each quarter (March 31, June 30, Sept 30, Dec 31)

Comprehensive quarterly deep audit for Green Man Tavern.

This is a full re-evaluation compared to the baseline and previous quarters.

PART 1: DATABASE HEALTH
- How much data in production? (users, systems, connections, conversations)
- Any orphaned records? (systems with no connections, users with no progress)
- Are backups happening and restorable?
- Migration history: Any failed or sketchy migrations?
- Data integrity: Any constraint violations?

PART 2: CODE QUALITY TRENDS
- Total lines of code (trending up/down/stable)?
- Cyclomatic complexity of key modules (getting simpler/more complex)?
- Which modules change most frequently? (candidates for instability)
- Which modules have ZERO tests? (risk areas)
- Code duplication: Any patterns repeated in 3+ places?

PART 3: TECHNICAL DEBT ASSESSMENT
- What's the biggest piece of tech debt?
- What should be refactored before it becomes urgent?
- Any deprecated libraries still being used?
- Any abandoned features cluttering the codebase?
- What would take most value to fix?

PART 4: PERFORMANCE TRENDS
- Median page load time? (goal: <2 seconds)
- Average MindsDB agent response time? (goal: <5 seconds)
- Database query performance? (any queries >500ms?)
- Memory usage? (any leaks detected?)
- PubSub message volume? (sustainable levels?)

PART 5: SECURITY POSTURE
- Any new vulnerability types introduced this quarter?
- Authentication mechanism: Still solid?
- Authorization checks: Consistently applied?
- Any new attack surfaces opened?
- User data handling: Any new risks?

PART 6: FEATURE COMPLETENESS
- Which of the "Seven Seekers" implementation is 100% done?
- Systems Flow Diagram: What percentage complete?
- Quest system: Implemented? Working?
- Achievement system: Implemented? Working?
- Opportunity detection: Implemented? Accurate?
- MindsDB integration: Stable? Any recurring issues?

PART 7: TESTING TRENDS
- Coverage percentage (target: >75%)?
- Test execution time (target: <30 seconds)?
- Any flaky tests?
- Are integration tests comprehensive?
- Are production issues catching in tests or in production?

PART 8: TEAM VELOCITY
- How many features shipped this quarter?
- How many bugs fixed?
- How many tech debt items addressed?
- Ratio: New features vs. bug fixes vs. tech debt?
- Is the team going faster or slower over time?

PART 9: USER-FACING QUALITY
- Any user-reported bugs?
- Feature adoption: Which features are actually used?
- Performance: Any user complaints about speed?
- Usability: Any confusing parts?
- Character agents: Are responses helpful or generic?

PART 10: ROADMAP ALIGNMENT
- Are we on track for planned features?
- Are priorities still correct, or should they shift?
- Any blockers preventing progress?
- Any new opportunities discovered?

DELIVERABLES:
1. Detailed comparison to previous quarter
2. Trend analysis (improving/stable/declining)
3. Top 3 priorities for next quarter
4. Top 3 risks to monitor
5. Recommendation: Ship now? Or continue dev?

FORMAT: Create both a detailed report and an executive summary (1 page)

🔒 SECURITY DEEP-DIVE: Authorization Audit

Run before any major release or when adding user-facing features

Security authorization audit for Green Man Tavern.

This is a CRITICAL audit - potential for data breaches.

SCENARIO TESTING:

Scenario 1: User A tries to access User B's data
- Can user_a view user_b's systems diagram? (SHOULD BE: NO)
- Can user_a view user_b's quest progress? (SHOULD BE: NO)
- Can user_a view user_b's conversation history? (SHOULD BE: NO)
- Can user_a edit user_b's profile? (SHOULD BE: NO)
- Can user_a view user_b's achievements? (SHOULD BE: NO)

For each "YES" answer: 🔴 CRITICAL - Fix immediately

Scenario 2: Unauthenticated user
- Can someone access /database without logging in? (SHOULD BE: NO)
- Can someone access /living-web without logging in? (SHOULD BE: NO)
- Can someone access /character/:id without logging in? (SHOULD BE: NO)
- Any endpoints accessible without authentication?

For each "YES": 🔴 CRITICAL - Add authentication gate

Scenario 3: Character agents access control
- Can agent access only current_user's data? (SHOULD BE: YES)
- Can agent see other users' systems? (SHOULD BE: NO)
- Can agent suggest changes to wrong user's data? (SHOULD BE: NO)

Scenario 4: Input validation
- Can user input code/scripts in text fields? (SHOULD BE: NO)
- Are all form submissions validated on server? (SHOULD BE: YES)
- Are file uploads (if any) validated for type/size? (SHOULD BE: YES)

Scenario 5: Session security
- Do sessions expire? (SHOULD BE: YES, after 24 hours)
- Are old sessions invalidated on logout? (SHOULD BE: YES)
- Can session tokens be guessed? (SHOULD BE: NO)
- Is CSRF protection on all forms? (SHOULD BE: YES)

REPORTING:
- List each scenario: PASS / ⚠️ WARNING / 🔴 FAIL
- For each FAIL: Describe the vulnerability
- For each WARNING: Suggest improvement
- Rate overall security: 🟢 Secure / 🟡 Acceptable / 🔴 Critical Issues

If any 🔴 found: "DO NOT DEPLOY until fixed"

📊 PERFORMANCE PROFILING: Find Bottlenecks

Run when users report slowness or before optimization work

Performance profiling and bottleneck detection for Green Man Tavern.

PART 1: Database Query Analysis
- Which database queries run most frequently?
- Which queries take the longest?
- Are there N+1 query patterns?
  Example: "Loop over 100 systems, query connections for each" = 101 queries!
  Example: "Loop over users, query each user's systems" = problem!
- Which queries could be preloaded with :include or preload()?
- Which queries should have indexes?

Show: Query pattern → How many times per page load → Time per query

PART 2: LiveView Socket Assigns
- What's stored in socket.assigns?
- How much data? (size in KB/MB?)
- Is it necessary on every page load?
- Should any be moved to temporary_assigns?
- Could any data be lazy-loaded?

PART 3: MindsDB Agent Performance
- Average agent response time?
- Min/max response times?
- Are responses being cached?
- What's the cache hit rate?
- Should cache TTL be adjusted?
- Are there slow PDFs in the knowledge base?

PART 4: LiveView Rendering
- How many LiveView updates per user action?
- Are there unnecessary re-renders?
- Should any parts use temporary_assigns?
- Are streams being used for long lists?

PART 5: Frontend Performance
- Page load time (goal: <2 seconds)?
- Time to interactive?
- Are there large CSS files?
- Are there large JavaScript bundles?
- Should anything be lazy-loaded?

RECOMMENDATIONS:
Ranked by impact (highest first):
- If fixed: Would save X ms per page load
- Implementation difficulty: Easy / Medium / Hard
- Risk of regression: Low / Medium / High

END WITH: "Recommend optimizing: [top 3 items]"

🧪 TEST COVERAGE GAP ANALYSIS

Run monthly or before major refactoring

Test coverage gap analysis for Green Man Tavern.

PART 1: Missing Context Tests
- List all Context modules
- For each: Does it have tests?
- Coverage percentage per Context
- Which Context has lowest coverage?

PART 2: Missing LiveView Tests
- List all LiveView modules
- For each: Does it have tests?
- Are happy path tests present?
- Are error cases tested?
- Are permissions tested (auth checks)?

PART 3: Missing Integration Tests
- Is there a test for complete user registration flow?
- Is there a test for adding systems + creating connections?
- Is there a test for character chat + MindsDB integration?
- Is there a test for quest acceptance + completion?
- Is there a test for logging in + out + data persistence?

PART 4: MindsDB Agent Testing
- Are agent calls mocked in tests?
- Are agent responses validated?
- Are character personalities tested?
- Are edge cases tested (agent timeout, invalid response)?

PART 5: Security Testing
- Is there a test verifying user_a can't access user_b's data?
- Is there a test for unauthenticated access being blocked?
- Is there a test for CSRF protection?
- Are there tests for SQL injection vulnerabilities?

PART 6: Database Testing
- Are migrations tested?
- Do seeds produce expected data?
- Are constraints enforced?
- Are foreign key relationships tested?

RECOMMENDATIONS:
- Most impactful test to add: [X]
- Most at-risk area (low coverage): [Y]
- Suggest: "Add these tests first: [list]"

GOAL: Get coverage above 75%

🚨 PRE-DEPLOYMENT CHECKLIST

Run 24 hours before any production deployment

Pre-deployment verification for Green Man Tavern.

This is go/no-go decision for deployment.

MANDATORY CHECKS:

☐ Authentication: Can't access protected pages without login?
☐ User Isolation: User A can't see User B's data?
☐ All Tests Pass: Any test failures?
☐ Database Migrations: All up to date? Rollback tested?
☐ Error Handling: All errors gracefully handled?
☐ Performance: Page loads in <2 seconds?
☐ MindsDB: Agent service healthy?
☐ Environment Variables: All production values set?
☐ Backups: Database backup recent and restorable?
☐ Monitoring: Error tracking enabled?
☐ Logging: Production logging active?

QUALITY GATES (any red = DO NOT DEPLOY):
- Test coverage: ✅ >75% / ⚠️ 60-75% / 🔴 <60%
- Critical issues: ✅ Zero / ⚠️ <3 High-priority / 🔴 3+
- Security issues: ✅ Zero / ⚠️ Minor / 🔴 Any critical
- Performance regressions: ✅ None / ⚠️ <10% / 🔴 >10%

RECOMMENDATION:
- ✅ SAFE TO DEPLOY
- ⚠️ DEPLOY WITH CAUTION (risks: [list])
- 🔴 DO NOT DEPLOY (blockers: [list])

If deploying: Document any known issues

🔍 REFACTORING GUIDANCE

Use when code feels messy but you're not sure where to start

Refactoring guidance for Green Man Tavern.

Analyze code quality and suggest refactoring priorities.

PART 1: Duplication Analysis
- Find code patterns repeated 3+ times
- For each: "Should this be extracted to a helper function?"
- Suggest: Extract to [module] as [function_name]

PART 2: Complexity Analysis
- Which functions are too complex (cyclomatic complexity >10)?
- Which modules are too large (>300 lines)?
- Which modules do too many things (violating single responsibility)?

Suggest: "This module should be split into [list]"

PART 3: Naming Issues
- Are there variables with unclear names?
- Are there functions that don't match what they do?
- Are there inconsistently named similar things?

PART 4: Dead Code
- Are there unused functions?
- Are there unused modules?
- Are there commented-out code blocks?
- Are there TODO/FIXME comments that are old?

PART 5: Anti-Patterns
- LiveViews doing database queries directly?
- Contexts mixing business logic with database logic?
- Large controller functions?
- Deep nesting in conditionals?
- Big if/else chains that should be pattern matching?

RECOMMENDATIONS (Ranked by impact + effort):
1. High impact, low effort: [suggestion]
2. Medium impact, low effort: [suggestion]
3. High impact, medium effort: [suggestion]

SUGGESTED REFACTORING PLAN:
"Start with #1, then #2, then #3"
"You could do this in [X hours]"

📝 Documentation Verification

Run before release or when onboarding new developer

Documentation completeness check for Green Man Tavern.

PART 1: Module Documentation
- How many modules have @moduledoc comments? (target: 100%)
- How many public functions have @doc comments? (target: 100%)
- Are complex algorithms explained with comments?
- Are edge cases documented?

PART 2: README & Setup
- Is there a README.md?
- Does it explain the project's purpose?
- Are setup instructions clear?
- Are there any broken links?
- Can a new developer follow it successfully?

PART 3: Architecture Documentation
- Is there an ARCHITECTURE.md explaining components?
- Is the Seven Seekers system documented?
- Is the Systems Flow Diagram documented?
- Are integration points documented?
- Is the database schema documented?

PART 4: API Documentation
- Are Context functions documented?
- Are LiveView events documented?
- Are data structures documented?
- Are error codes documented?

PART 5: Contributing Guide
- Is there a CONTRIBUTING.md?
- Are code standards documented?
- Are common tasks explained?
- Is the development workflow clear?

MISSING DOCUMENTATION (Most important first):
- [Item 1]: [Why it's important]
- [Item 2]: [Why it's important]

RECOMMENDATIONS:
"Before release: Add these docs: [list]"
"Nice to have: [list]"

🎯 CUSTOM: Check Specific Module

Use for targeted audits of specific components

```
Audit this specific module: [MODULE_NAME]

Analyze:

1. PURPOSE: What does this module do?
2. DEPENDENCIES: What other modules does it depend on?
3. USAGE: Where is this module used?
4. COMPLEXITY: Is it doing too much?
5. TESTING: What tests exist? Coverage?
6. DOCUMENTATION: Is it documented?
7. PERFORMANCE: Any bottlenecks?
8. SECURITY: Any data access concerns?
9. QUALITY: Code quality issues?
10. CONSISTENCY: Does it follow project patterns?

RECOMMENDATIONS:
- Is this module healthy or needs attention?
- Should it be refactored?
- Should it be split into smaller modules?
- Are there obvious bugs or issues?

END WITH: "Overall health: 🟢 Healthy / 🟡 Needs Attention / 🔴 Needs Work"
```

📤 HOW TO USE THESE PROMPTS

In Claude Code:

1. **Copy the entire prompt** (from `to`)
2. **Run**: `claude-code` then paste the prompt
3. **Claude Code analyzes your full codebase** and reports results
4. **Save the output** to `.claude/audit_results/[date]-[check-type].md`

Save Results Pattern:

```bash
# After running an audit:
claude-code "[PROMPT]" > .claude/audit_results/2025-01-20-weekly-health.md
```

Running on Schedule:

Add to your calendar:

- **Monday 9:00 AM**: Weekly Structural Health Check
- **Thursday 9:00 AM**: Bi-Weekly Architecture Drift
- **1st of month 9:00 AM**: Monthly Integration Test
- **Quarter end**: Deep Audit
- **Before deployment**: Pre-Deployment Checklist

Interpreting Results:

- 🟢 **Green**: All good, no action needed
- 🟡 **Yellow**: Monitor or plan improvement
- 🔴 **Red**: Fix before continuing development

✅ COMPLETION CHECKLIST

Use this to track which audits you've completed:

INITIAL SETUP (Week 1)

☐ Run Master Initial Audit

☐ Create .claude/project_standards.md

☐ Create AUDIT_BASELINE.md with results

☐ Create IMPLEMENTATION_LOG.md

☐ Create .claude/audit_results/ folder

☐ Review and prioritize findings

ONGOING (Weekly)

☐ Monday: Weekly Structural Health Check

☐ Thursday: Bi-Weekly Architecture Drift

☐ Friday: Fix any issues found

☐ End of week: Update IMPLEMENTATION_LOG.md

MILESTONE (Monthly/Quarterly)

☐ 1st: Full Integration Test

☐ Quarterly: Deep Audit

☐ Before deployment: Security + Performance audits

☐ Before deployment: Pre-Deployment Checklist

ARCHIVE

☐ Save each audit result to .claude/audit_results/[date].md

☐ Update AUDIT_BASELINE.md with new metrics

☐ Track trends in a spreadsheet or document

Version: 1.0

Last Updated: Today

Total Prompts: 13 (1 master + 6 ongoing + 6 special-purpose)