Test-Driven Implementation Skill

What This Skill Does

This skill prevents the #1 problem you encountered: **Claude forgetting to create necessary tables, missing dependencies, and skipping testing.**

It enforces a systematic 3-phase approach:

1. **Phase 1:** List ALL dependencies upfront (tables, APIs, components)
2. **Phase 2:** Test EACH component after creation
3. **Phase 3:** End-to-end validation with all user roles

How to Install

Option 1: Use in Claude Chat (Manual)

When starting a new feature implementation, give Claude this instruction:

```
Before implementing [feature name], read the test-driven-implementation skill
and follow its systematic approach.

Location: /mnt/skills/user/test-driven-implementation/SKILL.md
```

Claude will then:

- Read the skill first
- Follow the 3-phase approach
- List all dependencies before coding
- Test each step incrementally
- Run end-to-end validation

Option 2: Add to Your Skills Directory

If you want this skill to be automatically available:

1. Copy the `test-driven-implementation` folder to your project
2. Place it in: `/mnt/skills/user/test-driven-implementation/`
3. Claude will detect it automatically when relevant

What Problem This Solves

❌ Without This Skill:

Claude's typical behavior:

- Starts coding immediately
- Forgets about junction tables
- Doesn't test after each step
- Moves on before verification
- You discover missing tables hours later

Example from your project:

- Built invitation system
- Forgot `user_invitations` table
- Forgot `session_access` table
- Forgot `is_active` column
- Multiple rounds of fixes needed

✅ With This Skill:

Phase 1 forces analysis:

```markdown
Database Requirements:
- [ ] users table → EXISTS ✅
- [ ] user_invitations table → MISSING ❌
- [ ] session_access table → MISSING ❌
- [ ] is_active column → MISSING ❌

STOP: Need to create migrations 010-012 FIRST
```

Phase 2 forces testing:

```bash
Migration 010 applied
Verification query: SELECT user_role FROM users LIMIT 1;
Result: 'admin' ✅
Status: Safe to proceed
```

Phase 3 forces end-to-end:

```markdown
Test: Analyst invitation flow
1. Create invitation → ✅
2. Accept invitation → ✅
3. Login as analyst → ✅
4. Verify limited access → ✅
All tests pass → Feature complete
```

When to Use This Skill

Use for ANY feature involving:

- Database schema changes
- New API endpoints
- Role-based access control
- Multi-user functionality
- Permission systems
- Data isolation
- User authentication

Especially critical for:

- Adding new user roles
- Implementing access control
- Building invitation systems
- Any "admin vs user" distinction

Example Usage

Scenario: Building a Comments Feature

Without skill:

```
Claude: "I'll create a comments API endpoint..."
[Creates endpoint, forgets comments table]
You: "Error: table 'comments' does not exist"
Claude: "Oh right, let me create that..."
[Creates table, forgets user_id foreign key]
You: "Error: foreign key violation..."
```

With skill:

```
You: "Build a comments feature. Use test-driven-implementation skill."

Claude: "Reading /mnt/skills/user/test-driven-implementation/SKILL.md..."

Claude: "Phase 1: Pre-Implementation Analysis

Database Requirements:
- [ ] comments table (comment_id, submission_id, user_id, text, created_at)
- [ ] Foreign key: comments.submission_id → document_submissions.submission_id
- [ ] Foreign key: comments.user_id → users.user_id
- [ ] Index: idx_comments_submission (for fast lookup)
- [ ] Index: idx_comments_user (for filtering by user)

API Requirements:
- [ ] POST /submissions/:id/comments (permission: must have submission access)
- [ ] GET /submissions/:id/comments (permission: must have submission access)
- [ ] DELETE /comments/:id (permission: own comments only)

Frontend Requirements:
- [ ] CommentsList component
- [ ] CommentForm component
- [ ] API integration with error handling

Approve this plan before I proceed?"

You: "Approved"

Claude: "Creating Migration 015..."
[Creates migration]
"Testing migration..."
[Runs verification query]
"✅ Table created, columns verified, indexes created"

[Continues with systematic verification at each step]
```

What's Included

SKILL.md (5KB)

- Complete methodology
- Phase-by-phase instructions
- Verification checklists
- Common mistakes to avoid
- Success criteria

- Quick-start template

references/verification-queries.md (7KB)

- Pre-written SQL queries
- Copy-paste verification queries
- Data isolation checks
- Permission verification
- Performance checks
- Quick diagnostics

Benefits You'll See

Time Saved

- **Before:** 2-3 hours debugging missing dependencies
- **After:** 15 minutes upfront planning, no debugging

Quality Improved

- **Before:** Discover bugs during manual testing
- **After:** Catch 90% of bugs during incremental verification

Confidence

- **Before:** "Hope this works..."
- **After:** "Tested at each step, verified end-to-end"

Documentation

- **Before:** No record of what was tested
- **After:** Complete test log with results

Tips for Best Results

1. Reference the Skill Explicitly

```
"Build [feature]. Follow test-driven-implementation skill methodology."
```

2. Insist on Phase 1 Approval

```
"Don't start coding until I approve the dependency analysis."
```

3. Request Verification After Each Step

```
"Apply migration and show me the verification query results."
```

4. Demand End-to-End Tests

```
"Run all test scenarios from Phase 3 before marking complete."
```

Real-World Impact

Your Analyst Role Feature

Without skill:

- Multiple missing tables discovered during testing
- Several debugging sessions needed
- ~6-8 hours of fixes

With skill (estimated):

- All dependencies listed in Phase 1
- Tables created before APIs
- Incremental testing catches issues early
- ~2-3 hours total, first-time success

Future Skills to Create

Based on your experience, consider creating:

1. **aws-fullstack-deployment** - CDK, Lambda, RDS patterns
2. **database-migration-management** - Safe schema evolution
3. **rbac-implementation** - Role-based access patterns
4. **react-multi-role-ui** - Permission-aware components
5. **cost-monitoring** - Token tracking patterns

Each skill captures your hard-won knowledge!

Questions?

This skill is based on YOUR learning from the analyst role implementation. It captures the systematic approach you forced Claude to follow after discovering problems.

The skill will evolve as you use it and discover improvements!

Remember: The skill exists because AI tends to skip verification. Use it religiously for multi-step features!