Day 2: Interview-Driven TDD

AI Interviews You to Create Tests That Match Your Vision

Fleet Management API: Vehicles, Drivers & Assignments

1. Task Overview

Duration	5-7 hours
Difficulty	Intermediate to Advanced
Prerequisites	Day 1 SDD task completed, basic pytest knowledge
AI Tools	Claude Code, GitHub Copilot, or Google Gemini

🎯 Core Concept: Interview-Driven Test Development

Instead of AI making up tests, YOU define what correct behavior looks like through a structured interview. AI asks questions, you provide answers, and together you create functional tests that capture YOUR requirements. All other tests (unit, integration, E2E) are derived from these functional tests—ensuring perfect alignment.

Learning Objectives

1. Use AI interview mode to elicit and document YOUR exact requirements
2. Create functional tests through structured conversation with AI
3. Derive unit, integration, and E2E tests from approved functional tests
4. Ensure all tests trace back to user-validated requirements
5. Create AI Skill files that enforce your coding standards

📋 API Scope: Three Entities

Your Fleet Management API must support:

- **Vehicles** - Fleet vehicles (make, model, year, type, fuel_type, status)
- **Drivers** - People who operate vehicles (name, license, contact info, status)
- **Assignments** - Links drivers to vehicles for a time period (driver_id, vehicle_id, dates)

💡 **Tip:** *Use AI interviews to derive the exact schema for Drivers and Assignments based on your requirements!*

2. The Interview-Driven Approach

Traditional AI code generation has a fundamental problem: AI invents tests based on assumptions. This leads to tests that don't match what you actually want. The Interview-Driven approach flips this:

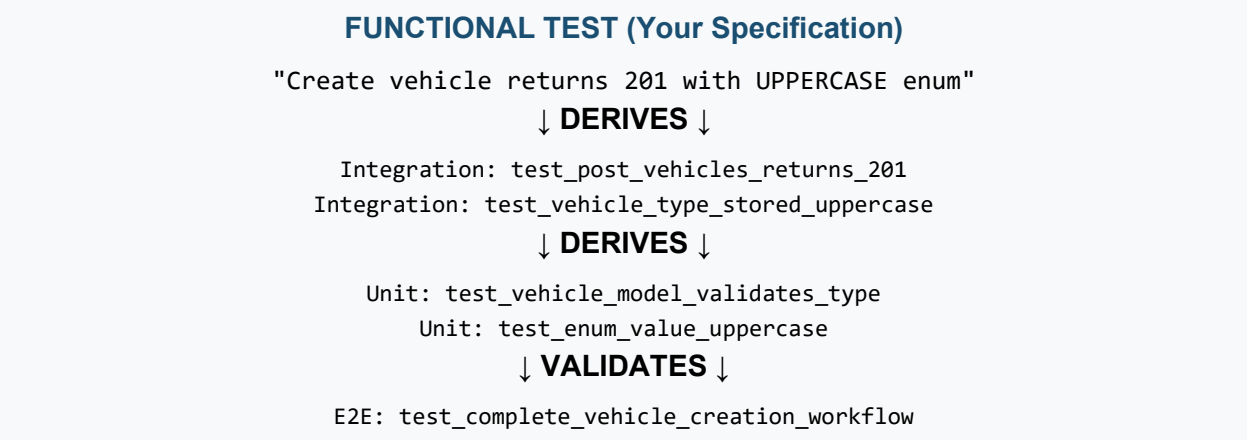
❌ Traditional AI Testing	✅ Interview-Driven Testing
You say: "Write tests for vehicle API"	AI asks: "What should happen when creating a vehicle?"
AI invents requirements	You define requirements through answers
Tests may not match your needs	Tests capture exactly what you specified
Constant rework and corrections	Right the first time

2.1 The Test Derivation Chain

🔗 The Golden Rule: All Tests Must Trace to Functional Tests

Functional Tests (from interview) → Integration Tests → Unit Tests → E2E Tests

AI cannot add new behaviors or requirements. Every test must trace back to something YOU approved in the functional tests. This prevents AI from "making things up."



3. Interview Tools by AI Platform

Each AI platform has different mechanisms for conducting structured interviews. Choose the approach that matches your tool:

3.1 Claude Code: AskUserQuestion Tool

Claude Code has a built-in interview mechanism called AskUserQuestion that presents structured questions with options:

```
# Start an interview session in Claude Code:

"Before writing any tests, interview me about the vehicle API requirements.
Use the AskUserQuestion tool to ask about:
1. What should POST /vehicles return on success?
2. What format should vehicle_type use (lowercase/UPPERCASE)?
3. What happens when required fields are missing?
4. How should the API handle duplicate vehicles?

Present options when possible. Don't proceed until I've answered."
```

Claude will ask questions one at a time, presenting multiple-choice options based on common patterns. Your answers become the specification.

3.2 GitHub Copilot Chat: Conversational Interview

Copilot Chat supports multi-turn conversations. Use this prompt to initiate interview mode:

```
# In Copilot Chat (VS Code or GitHub):

"I want you to interview me to understand my vehicle API requirements.
Ask me ONE question at a time about:
- Expected HTTP status codes for each operation
- Data validation rules
- Error response formats
- Edge cases I care about

After each answer, summarize what you understood and ask the next question.
Don't generate any code until the interview is complete."
```

Copilot will engage in back-and-forth conversation. Keep answers short and specific.

3.3 Google Gemini: Structured Chat Interview

Gemini (in Gems or standard chat) can be prompted for interview-style conversations:

```
# In Gemini Chat or create a Gem with these instructions:

"You are a requirements analyst. Your job is to interview me about
my FastAPI vehicle API before any tests are written.

Interview structure:
1. Ask about the user story (who uses this, why)
2. Ask about happy path behavior
3. Ask about error conditions
4. Ask about edge cases
5. Summarize all requirements in Given-When-Then format

Ask ONE question, wait for my answer, then proceed.
Never assume - always ask."
```

3.4 Interview Tool Comparison

Feature	Claude Code	Copilot Chat	Gemini
Interview Tool	AskUserQuestion (built-in)	Chat conversation	Chat / Gems
Multiple Choice	Yes (native)	Via prompting	Via prompting
Session Memory	Full context	Conversation thread	Conversation thread
Best For	Structured spec building	IDE-integrated interviews	Flexible conversation

4. The Interview Process: From Questions to Tests

4.1 Phase 1: User Story Interview (15-20 min per feature)

AI asks questions to understand the user story. You provide answers that become your specification. Do this for each entity (Vehicles, Drivers, Assignments):

Example Interview: Create Vehicle Feature

AI: *"Who will use this endpoint and what's their goal?"*

You: "Fleet managers adding new vehicles to track in their system."

AI: *"What data is required when creating a vehicle?"*

You: "Make, model, year, vehicle_type, and fuel_type. All required."

AI: *"What HTTP status should be returned on success?"*

You: "201 Created, with the new vehicle including its generated ID."

AI: *"How should vehicle_type be stored - lowercase or UPPERCASE?"*

You: "UPPERCASE always. Like SEDAN, SUV, TRUCK."

4.2 Phase 2: Convert Answers to Functional Tests

After the interview, AI generates functional tests based ONLY on your answers. You review and approve:

```
# tests/functional/test_vehicle_workflows.py
# Generated from interview - APPROVED BY USER


class TestCreateVehicleWorkflow:
    """
    User Story: Fleet managers adding new vehicles to track.
    Source: Interview 2024-01-15, answers 1-4
    """


    def test_create_vehicle_complete_workflow(self, client, db):
        # From interview: All fields required
        vehicle_data = {
            "make": "Tesla",
            "model": "Model 3",
            "year": 2024,
            "vehicle_type": "SEDAN",  # From interview: UPPERCASE
            "fuel_type": "ELECTRIC"
        }


        response = client.post("/vehicles", json=vehicle_data)
```

```
        # From interview: 201 Created with generated ID
        assert response.status_code == 201
        assert "id" in response.json()
        assert response.json()["vehicle_type"] == "SEDAN"  # UPPERCASE
```

4.2.1 Deriving Entity Schemas with AI

For Drivers and Assignments, use AI interviews to derive the schema. Let AI ask questions about what data you need:

> **🔧 Schema Derivation Interview Example**
>
> **Prompt:** *"Interview me to derive the schema for the Driver entity. Ask about required fields, data types, validation rules, and relationships."*
>
> **AI:** *"What information do you need to identify a driver?"*
>
> **You:** "Name, email, phone, and driver's license number."
>
> **AI:** *"Should license number be unique? What format?"*
>
> **You:** "Yes unique. Format varies by state, so just validate it's not empty."
>
> **AI:** *"What statuses can a driver have?"*
>
> **You:** "ACTIVE, INACTIVE, SUSPENDED. UPPERCASE enum like vehicles."

> **🔗 Assignment Entity Interview**
>
> **Prompt:** *"Interview me about Assignments - the entity that links Drivers to Vehicles."*
>
> **Key questions AI should ask:**
>
> - Can a driver have multiple active assignments?
> - Can a vehicle have multiple drivers assigned?
> - What date fields are needed (start_date, end_date)?
> - What happens if driver or vehicle is deleted?
> - What assignment statuses exist (ACTIVE, COMPLETED, CANCELLED)?

4.3 Phase 3: Derive Other Tests (AI-Assisted)

With approved functional tests, instruct AI to derive other test types. AI must reference the functional test:

```
# Prompt to AI:
"From the approved functional test 'test_create_vehicle_complete_workflow',
derive the following tests. Each test MUST trace to a specific assertion
in the functional test:

1. Integration tests for POST /vehicles endpoint
2. Unit tests for the Vehicle model validation
3. E2E test for vehicle creation with real database
```

```
Include a comment showing which functional test assertion each derives from."
```

```
# AI generates with traceability comments:

# tests/integration/test_vehicles_api.py
class TestCreateVehicle:
    # Derived from: test_create_vehicle_complete_workflow

    def test_post_returns_201(self, test_client):
        # Traces to: assert response.status_code == 201
        response = test_client.post("/vehicles", json=valid_data)
        assert response.status_code == 201

    def test_response_includes_id(self, test_client):
        # Traces to: assert "id" in response.json()
        response = test_client.post("/vehicles", json=valid_data)
        assert "id" in response.json()

    def test_vehicle_type_uppercase(self, test_client):
        # Traces to: assert response.json()["vehicle_type"] == "SEDAN"
        response = test_client.post("/vehicles", json=valid_data)
        assert response.json()["vehicle_type"] == "SEDAN"
```

5. AI Skill Files: Encoding Your Standards

While interviews capture WHAT to test, Skill files capture HOW to write code. They ensure AI follows your conventions:

5.1 Skill File Structure

Tool	File Location	Format	Auto-Discovery
Claude Code	.claude/skills/<name>/SKILL.md	YAML frontmatter + Markdown	Yes (by description)
Copilot	.github/copilot-instructions.md	Markdown	Yes (always loaded)
Gemini	Gems UI / Chat context	Plain text instructions	Manual activation

5.2 Example Skill File (Claude Code)

```
---
name: fleet-management-api
description: Fleet Management API with Vehicles, Drivers, and Assignments.
  Use when creating endpoints, models, or tests for the fleet API.
---

# Fleet Management API Development Standards

## Entities
- Vehicles: Fleet vehicles with type, fuel, status
- Drivers: People who operate vehicles
- Assignments: Links drivers to vehicles for time periods

## Technology Stack
- FastAPI for REST API
- pymongo (NOT motor) for MongoDB
- pytest with pytest-cov for testing

## Conventions
- Enum values: UPPERCASE (SEDAN, ACTIVE, SUSPENDED)
- HTTP 201 for resource creation
- HTTP 404 with {"detail": "message"} for not found
- UUID strings for all IDs
- Foreign key validation on Assignments
```

```
## Test Requirements
- All tests must trace to a functional test
- Include traceability comment on each test
- Minimum 90% coverage
```

6. Complete Workflow: Step by Step

Step	Phase	Actions
1	**Create Skill File**	Document technology choices, naming conventions, patterns (30 min)
2	**Start Interview**	Prompt AI to enter interview mode for first feature (5 min)
3	**Answer Questions**	AI asks about user story, behavior, errors, edge cases. You answer. (15-20 min)
4	**Review Functional Tests**	AI generates tests from your answers. You approve or request changes. (10 min)
5	**Derive Other Tests**	AI derives unit/integration/E2E tests with traceability comments. (15 min)
6	**Generate Code**	AI implements code to make YOUR tests pass, following Skill file. (30-45 min)
7	**Verify & Iterate**	Run tests, verify coverage, update Skill file with learnings. (15 min)
8	**Repeat**	Return to Step 2 for next feature. (~2 hours per feature)

6.1 Interview Prompts by Entity

Conduct interviews for each entity. Here are suggested prompts:

VEHICLES

Feature	Interview Prompt
Create Vehicle	"Interview me about POST /vehicles: required fields, success response, validation rules"
List/Get Vehicles	"Interview me about GET /vehicles: pagination, filtering by status/type, empty results"
Update/Delete	"Interview me about PUT/DELETE /vehicles/{id}: what's updateable, soft vs hard delete"

DRIVERS

Feature	Interview Prompt
Schema Design	"Interview me to derive the Driver schema: required fields, license validation, statuses"
Create Driver	"Interview me about POST /drivers: unique constraints, required contact info"

Driver Status	"Interview me about driver statuses: ACTIVE/INACTIVE/SUSPENDED transitions"

ASSIGNMENTS

Feature	Interview Prompt
Schema Design	"Interview me to derive Assignment schema: driver/vehicle links, date ranges, constraints"
Create Assignment	"Interview me about POST /assignments: FK validation, date overlap rules, active limits"
Assignment Rules	"Interview me about business rules: can driver have multiple? can vehicle have multiple?"
End Assignment	"Interview me about completing/cancelling assignments: status transitions, cascading"

7. Test Traceability Matrix

Every test must trace to a functional test. Document this traceability:

Functional Test	Derived Tests	Test Type

`test_create_vehicle_workflow`
Status: 201, ID generated, UPPERCASE enum
	`test_post_returns_201`
	`test_response_has_id`
	`test_type_uppercase`
		Integration
		Integration
		Unit

`test_create_driver_workflow` License unique, status ACTIVE by default	`test_driver_license_unique` `test_driver_default_active`	Integration Unit
`test_create_assignment_workflow` FK validation, no date overlap	`test_assignment_validates_fk` `test_no_overlapping_dates`	Integration Integration
`test_get_vehicle_not_found` Status: 404, detail message	`test_get_returns_404` `test_404_has_detail`	Integration Unit

⚠ VALIDATION RULE

If AI generates a test that doesn't trace to an approved functional test, REJECT IT. Ask: "Which functional test does this derive from?" If AI cannot answer, the test is inventing requirements.

8. Deliverables

#	Deliverable	Requirements
1	**AI Skill File**	SKILL.md or copilot-instructions.md with project specifications
2	**Interview Transcripts**	Saved conversations showing AI questions and your answers for each feature
3	**Functional Tests**	tests/functional/ with user-approved tests for all 3 entities (min 9 tests)
4	**Integration Tests**	tests/integration/ with traceability comments to functional tests
5	**Unit Tests**	tests/unit/ with traceability comments to functional tests
6	**E2E Tests**	tests/e2e/ with real database, min 4 scenarios, traceability comments
7	**Traceability Matrix**	Document showing how every test traces to a functional test
8	**Coverage Report**	≥90% line coverage with pytest-cov
9	**Git History**	Commits showing: interview → functional tests → derived tests → code

9. Scoring Rubric (100 points, 80+ to pass)

Criterion	Points	Requirements for Full Credit
Interview Quality	20	Complete transcripts, clear answers, all features covered
Functional Tests	20	9+ tests (3 per entity) from interviews, capture user requirements
Test Traceability	15	Every derived test has traceability comment to functional test
E2E Tests	15	4+ scenarios with real DB, proper cleanup, traceability
AI Skill File	10	Complete specifications, correct format for chosen tool
Test Coverage (≥90%)	10	Line coverage ≥90%, all tests meaningful
Integration/Unit Tests	5	Properly derived from functional tests
Git History	5	Shows interview → tests → code progression
TOTAL	**100**	**Passing: 80+ points**

Remember the Core Principle

1. AI interviews YOU to understand requirements
2. YOU approve functional tests before any derivation
3. ALL other tests must trace back to approved functional tests
4. AI CANNOT add new behaviors—only derive from what you specified

Good luck! The interview approach ensures that your tests capture YOUR requirements, not AI assumptions.