Complete Code Design Prompt Template (Full Version)

Complete Implementation Guide Request

My Requirement

Describe what you want to build in **2–5 sentences**. Clearly include: - What problem it solves - What triggers it (user action, scheduled job, event-based, webhook, etc.) - What should happen as a result

Example:
"I want to build a database change monitoring system. When any table in PostgreSQL has INSERT/UPDATE/DELETE, the system should detect it automatically and call registered handler functions. Handlers can send notifications, invalidate cache, or sync data to external systems."

SECTION 1: PROBLEM UNDERSTANDING

1. What is the **core problem** being solved?
2. What is the **TRIGGER** (what starts the process)?
3. What is the **SOURCE of data**?
4. What is the **DESTINATION / outcome**?
5. List **all use cases** (primary + edge cases).

SECTION 2: DATA FLOW DIAGRAM (DFD)

Provide a **complete data flow** showing how data moves.

Required details: 1. External Source 2. Entry Point (first code) 3. Data Transformations 4. Component Interactions 5. Final Output

Format:

```
[Source]
   ↓ (what data?)
[Component A]
   ↓ (what data?)
[Component B]
   ↓ (what data?)
[Component C]
   ↓
[Final Output]
```

SECTION 3: COMPONENT IDENTIFICATION

For **every component**, provide:

- Name
- Type (class / function / enum / dataclass)
- Purpose (one line)
- Why this type (why class vs function?)
- Data it **receives** (input)
- Data it **produces** (output)
- Components it **depends on / uses**

SECTION 4: DATA STRUCTURES

For each data structure:

1. What data does it hold?
2. Where does this data **originate**?
3. Who **creates** it?
4. Who **uses** it?
5. Sample realistic data

SECTION 5: DETAILED COMPONENT DESIGN

For Each CLASS

CLASS: `ClassName`
Purpose: One-line description

Why CLASS (not function): - Reason 1 - Reason 2

DATA IT RECEIVES: - From where: - Format: - Example:

DATA IT PRODUCES: - Goes to: - Format: - Example:

ATTRIBUTES: | Attribute | Type | Why Needed | |---------|------|------------| | name | type | explanation |

METHODS: | Method | Parameters | Returns | Purpose | |-------|------------|---------|---------|

INIT DESIGN: - Parameter → why needed - What is initialized and why

EACH METHOD DETAIL: - INPUT - OUTPUT - LOGIC STEPS (numbered) - COMPONENTS USED

CODE SKELETON:

```python
class ClassName:
    def __init__(self):
        pass
```

HOW TO USE:

```
obj = ClassName()
```

For Each FUNCTION

FUNCTION: `function_name`

Purpose: One line

Why FUNCTION (not class): - Reason 1 - Reason 2

CALLED BY:

CALLS:

INPUTS: - param: type → source

OUTPUTS: - Return value - Side effects

LOGIC STEPS: 1. Step one 2. Step two

ERROR HANDLING: - Possible failures - Handling strategy

CODE SKELETON:

```python
def function_name(param):
    pass
```

For Each ENUM

ENUM: `EnumName`

Purpose: Why fixed values are required

VALUES: - VALUE = "value" → when to use

Why ENUM (not strings):

USED BY:

USAGE EXAMPLE:

```
EnumName.VALUE
```

For Each DATACLASS

DATACLASS: `DataName`

Purpose: What data it represents

Why DATACLASS:

FIELDS: | Field | Type | Source | |------|------|--------|

CREATED BY:

USED BY:

SAMPLE DATA:

```
{}
```

HOW TO CREATE / USE:

```
DataName(...)
```

SECTION 6: COMPLETE DATA FLOW (STEP-BY-STEP)

STEP 1: Trigger / Source
- What happens - Data format - Example

↓

STEP 2: Component A
- Receives - Processes - Produces

↓

FINAL STEP: Output / Result

SECTION 7: COMPONENT INTERACTION DIAGRAM

```
[Entry Point]
    ↓ calls
[Component A] ──creates──► [Data X]
    ↓ passes
[Component B]
    ↓ calls
[Component C]
    ↓ produces
[Final Output]
```

SECTION 8: WHERE DATA ORIGINATES (Source Mapping)

Data Field	Original Source	How Obtained
field	source	method

SECTION 9: FILE STRUCTURE & ORGANIZATION

```
project/
├── file1.py   # purpose
├── module/
│   ├── file2.py
│   └── file3.py
```

Why this structure: - Reason 1 - Reason 2

SECTION 10: IMPLEMENTATION ORDER (Dependencies)

Order	Component	Depends On	Why
1	Component A	None	Base
2	Component B	A	Uses A

SECTION 11: COMPLETE CODE SKELETON

- Full project-level skeleton
- Type hints
- Error handling

- Example usage

SECTION 12: THREADING / ASYNC ANALYSIS

1. Blocking operations
2. Background tasks
3. Thread / async model
4. Shared data + locking
5. Threading diagram

SECTION 13: ERROR HANDLING STRATEGY

For each component: - What can fail - How to handle - Recovery strategy

---

SECTION 14: TESTING APPROACH

For each component: - Unit test strategy - Sample test cases - Mocking requirements

---

SECTION 15: BEST PRACTICES & PITFALLS

DO: - Best practice 1 - Best practice 2

DON'T: - Common mistake 1 - Common mistake 2

SECTION 16: EXTENSION GUIDE

- Adding new handler
- Adding new data type
- Adding new source

Summary Checklist

1. Problem Understanding
2. Data Flow Diagram
3. Component Identification
4. Data Origin Tracing
5. Detailed Component Design
6. Component Interactions
7. File Structure
8. Implementation Order
9. Code Skeletons
10. Best Practices