AI AGENT SYSTEM

Production-Ready Template Architecture

Phase 1.5 Complete - Template Hardened and Verified

Last Updated: February 2026

PROJECT OVERVIEW

This is a production-ready template system for creating isolated AI agent projects. Each project is a standalone expert system with clean layer separation, immutable templates, and zero cross-project contamination.

System Design

Feature	**Description**
Single Domain Experts	PineScript, Python, Shopify, etc.
Multi-Agent Expansion	Multiple agents within one project
Multi-Project Portfolio	Unlimited isolated projects
Future Ready	Distributed deployment capable

ARCHITECTURE LAYERS

Layer	**Path**	**Responsibility**
Agent	/agents	Intelligence only
Controller	/controllers	Orchestration only
Tool	/tools	External access only
Memory	/memory	Storage abstraction only
Config	/configs	Behavior control only
Interface	/interfaces	UI only
Script	/scripts	Lifecycle only

LAYER RULES - DO NOT VIOLATE

Agents Layer

ALLOWED: Intelligence only

NOT ALLOWED: Database access, HTTP requests, File system operations, UI code, Orchestration logic

Controllers Layer

ALLOWED: Orchestration only

NOT ALLOWED: UI code, Embedding generation, Crawling logic, Direct database schema access

Tools Layer

ALLOWED: External access only

NOT ALLOWED: Intelligence logic, Business decisions

Memory Layer

ALLOWED: Storage abstraction only

NOT ALLOWED: Intelligence logic, Business decisions

Config Layer

ALLOWED: Configuration only

NOT ALLOWED: Hardcoded values in Python logic

Interface Layer

ALLOWED: UI only

NOT ALLOWED: Business logic, Database logic, Orchestration

Scripts Layer

ALLOWED: Lifecycle operations only

NOT ALLOWED: Runtime logic, Business logic

DIRECTORY STRUCTURE

```
~/agents/
│
├── templates/
│   └── project_template/
│       ├── agents/
│       │   └── base.py
│       ├── controllers/
│       │   └── controller.py
│       ├── tools/
│       │   ├── __init__.py
│       │   ├── database.py
│       │   └── web.py
│       ├── memory/
│       │   └── memory.py
│       ├── configs/
│       │   └── project.yaml
│       ├── interfaces/
│       │   └── streamlit_app.py
│       ├── scripts/
│       │   └── init_db.py
│       ├── docker-compose.yml
│       ├── .env.example
│       ├── .gitignore
│       ├── requirements.txt
│       └── README.md
│
├── projects/
│   ├── pinescript-expert/
│   ├── python-expert/
│   └── shopify-expert/
│
└── create_project.py
```

TEMPLATE RULES - CRITICAL

Rule 1: Never modify the template directly

Always work in generated projects. The template is immutable.

Rule 2: Never add domain packages to template requirements

Add domain-specific packages to project requirements after generation.

Rule 3: Never commit .env files

Only commit .env.example. Real API keys stay local.

Rule 4: Never hardcode ports

Always use ${PORT} placeholder in templates.

Rule 5: Never hardcode paths

Always use config or environment variables.

Rule 6: Each project gets unique port

Never reuse ports between projects.

Rule 7: Each project is completely isolated

No cross-project imports. No shared databases. No shared API keys.

FILE MANIFEST - 14 FILES

File 1: docker-compose.yml

Purpose: PostgreSQL with pgvector extension. Each project gets isolated database on unique port.

```
version: '3.8'
services:
  postgres:
    image: pgvector/pgvector:pg16
    platform: linux/amd64
    container_name: ${PROJECT_NAME}_pgvector
    environment:
      POSTGRES_PASSWORD: postgres
      POSTGRES_USER: postgres
      POSTGRES_DB: ${PROJECT_NAME}
    ports:
      - "${PORT}:5432"
    volumes:
      - ./postgres_data:/var/lib/postgresql/data
    restart: unless-stopped
```

File 2: .env.example

Purpose: Environment variable template. Copy to .env and add real values.

```
PROJECT_NAME=your_project_name
PORT=54322
OPENAI_API_KEY=sk-...
DATABASE_URL=postgresql://postgres:postgres@localhost:${PORT}/${PROJECT_NAME}
```

File 3: requirements.txt

Purpose: Core dependencies only. No domain-specific packages.

```
openai
psycopg2-binary
streamlit
python-dotenv
pyyaml
pydantic-ai
```

File 4: README.md

Purpose: Project documentation. {PROJECT_NAME} replaced by generator.

```
# {PROJECT_NAME} Expert

## Architecture Layers

| Layer | Path | Responsibility |
|-------|------|----------------|
| Agent | /agents | Intelligence only |
| Controller | /controllers | Orchestration only |
| Tool | /tools | External access only |
| Memory | /memory | Storage abstraction only |
| Config | /configs | Behavior control only |
| Interface | /interfaces | UI only |
| Script | /scripts | Lifecycle only |

## Quick Start

cp .env.example .env
docker-compose up -d
pip install -r requirements.txt
python scripts/init_db.py
streamlit run interfaces/streamlit_app.py
```

File 5: agents/base.py

Purpose: Base class for all agents. Pure intelligence, no external access.

```
from pydantic_ai import Agent
from dotenv import load_dotenv

class BaseAgent:
    def __init__(self, name, system_prompt, config=None):
        load_dotenv()
        self.name = name
        self.config = config or {}
        model = self.config.get('model', 'openai:gpt-4o-mini')
        temperature = self.config.get('temperature', 0.2)

        self.agent = Agent(
            model,
            system_prompt=system_prompt,
            model_settings={'temperature': temperature}
        )

    async def run(self, query):
        result = await self.agent.run(query)
        return result.data
```

File 6: controllers/controller.py

Purpose: Orchestration layer. Load → Decide → Act → Think → Remember pattern.

```
class Controller:
    def __init__(self, agent, memory, tools, config):
        self.agent = agent
        self.memory = memory
        self.tools = tools
        self.config = config

    async def process_query(self, query, session_id=None):
        context = await self._load_context(session_id, query)
        plan = await self._decide_plan(query, context)
        tool_results = await self._execute_tools(plan)
        response = await self._think(query, context, tool_results)
        await self._remember(session_id, query, response)
        return response

    async def _load_context(self, session_id, query):
        return {}

    async def _decide_plan(self, query, context):
        return {"action": "direct_response", "tools": []}

    async def _execute_tools(self, plan):
        return {}

    async def _think(self, query, context, tool_results):
        return await self.agent.run(query)

    async def _remember(self, session_id, query, response):
        pass
```

File 7: tools/__init__.py

Purpose: Tool layer exports.

```
from .database import DatabaseTool
from .web import WebTool

__all__ = ['DatabaseTool', 'WebTool']
```

File 8: tools/database.py

Purpose: Database abstraction. All database operations use this class.

```
import psycopg2
from psycopg2.extras import RealDictCursor
from dotenv import load_dotenv
import os

class DatabaseTool:
    def __init__(self):
        load_dotenv()
        self.connection_string = os.getenv('DATABASE_URL')
        self.conn = None

    def connect(self):
        if not self.conn or self.conn.closed:
            self.conn = psycopg2.connect(self.connection_string)
        return self.conn

    def execute(self, query, params=None):
        conn = self.connect()
        cur = conn.cursor()
        try:
            cur.execute(query, params or ())
            conn.commit()
            rowcount = cur.rowcount
        except Exception as e:
            conn.rollback()
            raise e
        finally:
            cur.close()
        return rowcount

    def query(self, query, params=None):
        conn = self.connect()
        cur = conn.cursor(cursor_factory=RealDictCursor)
        try:
            cur.execute(query, params or ())
            results = cur.fetchall()
        finally:
            cur.close()
        return results

    def close(self):
        if self.conn and not self.conn.closed:
            self.conn.close()
            self.conn = None
```

File 9: tools/web.py

Purpose: HTTP client abstraction. All web requests use this class.

```
import requests

class WebTool:
    def __init__(self):
        self.session = requests.Session()
        self.session.headers.update({
            'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)'
        })

    def get(self, url, timeout=10):
        response = self.session.get(url, timeout=timeout)
        response.raise_for_status()
        return response.text

    def close(self):
        self.session.close()
```

File 10: memory/memory.py

Purpose: Memory abstraction. All storage operations use this class.

```python
import json
import os
from datetime import datetime

class Memory:
    def __init__(self, namespace='default', config=None):
        self.namespace = namespace
        self.config = config or {}

        base_path = self.config.get('file_path', 'knowledge/memory')
        self.storage_path = f"{base_path}/{namespace}"
        os.makedirs(self.storage_path, exist_ok=True)

    def save_value(self, key, value):
        path = f'{self.storage_path}/{key}.json'
        with open(path, 'w') as f:
            json.dump({
                'value': value,
                'timestamp': datetime.now().isoformat()
            }, f)

    def load_value(self, key):
        path = f'{self.storage_path}/{key}.json'
        if os.path.exists(path):
            with open(path, 'r') as f:
                return json.load(f)
        return None

    def list_keys(self):
        return [f.replace('.json', '') for f in os.listdir(self.storage_path)]

    async def retrieve_context(self, session_id, query):
        return self.load_value(f"{session_id}_context")

    async def store_conversation(self, session_id, query, response):
        self.save_value(f"{session_id}_conversation", {
            'query': query,
            'response': response,
            'timestamp': datetime.now().isoformat()
        })
```

File 11: configs/project.yaml

Purpose: Central configuration. All behavior controlled here. ${PROJECT_NAME} replaced by generator.

```
project:
  name: ${PROJECT_NAME}
  version: 1.0.0
  environment: development

agent:
  default_model: openai:gpt-4o-mini
  default_temperature: 0.2
  max_tokens: 2000
  timeout_seconds: 30
  system_prompt: You are a helpful AI assistant.

memory:
  mode: file
  namespace: default
  file_path: knowledge/memory
  vector_dimension: 1536

logging:
  level: INFO
  format: json
  output: logs/

ui:
  theme: light
  page_title: AI Expert
  page_icon: 🤖

features:
  streaming: false
  citations: true
  debug_mode: false
```

File 12: interfaces/streamlit_app.py

Purpose: User interface. Web-based chat. UI only, calls controller.

```
import streamlit as st
import sys
import os
import yaml
import asyncio
from pathlib import Path

sys.path.append(os.path.dirname(os.path.dirname(os.path.abspath(__file__))))

config_path = Path(__file__).parent.parent / 'configs' / 'project.yaml'
with open(config_path, 'r') as f:
    config = yaml.safe_load(f)

st.set_page_config(
    page_title=config['ui']['page_title'],
    page_icon=config['ui']['page_icon'],
    layout='centered'
)

st.title(f"{config['ui']['page_icon']} {config['project']['name']}")

from agents.base import BaseAgent
from controllers.controller import Controller
from memory.memory import Memory
from tools import DatabaseTool, WebTool

if 'controller' not in st.session_state:
    agent = BaseAgent(
        name='expert',
        system_prompt=f"You are an expert in {config['project']['name']}. Provide
clear, accurate, and helpful responses.",
        config=config['agent']
    )
    memory = Memory(
        namespace=config['memory']['namespace'],
        config=config['memory']
    )
    tools = {
        'db': DatabaseTool(),
        'web': WebTool()
    }
    st.session_state.controller = Controller(agent, memory, tools, config)

if 'messages' not in st.session_state:
    st.session_state.messages = []

def run_async(coro):
    try:
        loop = asyncio.get_event_loop()
    except RuntimeError:
        loop = asyncio.new_event_loop()
        asyncio.set_event_loop(loop)

    try:
        return loop.run_until_complete(coro)
    except Exception as e:
        st.error(f"Error: {str(e)}")
        return f"An error occurred: {str(e)}"
```

```
for message in st.session_state.messages:
    with st.chat_message(message['role']):
        st.markdown(message['content'])

if prompt := st.chat_input('Ask me anything...'):
    st.session_state.messages.append({'role': 'user', 'content': prompt})
    with st.chat_message('user'):
        st.markdown(prompt)

    with st.chat_message('assistant'):
        with st.spinner('Thinking...'):
            response = run_async(
                st.session_state.controller.process_query(prompt)
            )
            st.markdown(response)
            st.session_state.messages.append(
                {'role': 'assistant', 'content': response}
            )
```

File 13: scripts/init_db.py

Purpose: Database initialization. Run once per project after Docker starts.

```
import sys
import os
sys.path.append(os.path.dirname(os.path.dirname(os.path.abspath(__file__))))

from tools.database import DatabaseTool

def init_database():
    db = DatabaseTool()

    db.execute('CREATE EXTENSION IF NOT EXISTS vector')

    db.execute("""
        CREATE TABLE IF NOT EXISTS documentation (
            id SERIAL PRIMARY KEY,
            title TEXT,
            url TEXT UNIQUE,
            content TEXT,
            embedding vector(1536),
            created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
        )
    """)

    db.execute("""
        CREATE TABLE IF NOT EXISTS memory (
            id SERIAL PRIMARY KEY,
            namespace TEXT,
            key TEXT,
            value JSONB,
            embedding vector(1536),
            created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
            UNIQUE(namespace, key)
        )
    """)

    db.close()
    print("Database initialized successfully")

if __name__ == '__main__':
    init_database()
```

File 14: .gitignore

Purpose: Prevent committing sensitive or generated files.

```
.env
postgres_data/
logs/
knowledge/
__pycache__/
*.pyc
*.pyo
*.pyd
.Python
*.so
*.egg
*.egg-info/
dist/
build/
.DS_Store
.vscode/
.idea/
*.swp
*.swo
*~
```

PROJECT GENERATOR

Location: ~/agents/create_project.py

Purpose: Creates new isolated projects from immutable template.

```
import os
import shutil
import sys

def create_project(domain, port=54322):
    home = os.path.expanduser("~")
    project_name = f"{domain}-expert"
    project_path = f"{home}/agents/projects/{project_name}"
    template_path = f"{home}/agents/templates/project_template"

    print(f"Creating {project_name} on port {port}")

    if not os.path.exists(template_path):
        print(f"Error: Template not found at {template_path}")
        sys.exit(1)

    os.makedirs(f"{home}/agents/projects", exist_ok=True)
    shutil.copytree(template_path, project_path)

    shutil.copy(
        f"{project_path}/.env.example",
        f"{project_path}/.env"
    )

    with open(f"{project_path}/.env", 'r') as f:
        env_content = f.read()

    env_content = env_content.replace('your_project_name', domain)
    env_content = env_content.replace('54322', str(port))
    env_content = env_content.replace('${PORT}', str(port))
    env_content = env_content.replace('${PROJECT_NAME}', domain)

    with open(f"{project_path}/.env", 'w') as f:
        f.write(env_content)

    with open(f"{project_path}/README.md", 'r') as f:
        readme = f.read()
    readme = readme.replace('{PROJECT_NAME}', domain.title())
    with open(f"{project_path}/README.md", 'w') as f:
        f.write(readme)

    config_path = f"{project_path}/configs/project.yaml"
    with open(config_path, 'r') as f:
        config = f.read()
    config = config.replace('${PROJECT_NAME}', domain)
    with open(config_path, 'w') as f:
        f.write(config)

    os.makedirs(f"{project_path}/postgres_data", exist_ok=True)
    os.makedirs(f"{project_path}/knowledge", exist_ok=True)
    os.makedirs(f"{project_path}/knowledge/memory", exist_ok=True)
    os.makedirs(f"{project_path}/logs", exist_ok=True)

    print(f"Project created at: {project_path}")
    print("\nNEXT STEPS:")
    print(f"  cd ~/agents/projects/{project_name}")
    print("  # Add your OpenAI API key to .env")
    print("  docker-compose up -d")
    print("  pip install -r requirements.txt")
    print("  python scripts/init_db.py")
    print("  streamlit run interfaces/streamlit_app.py")

if __name__ == "__main__":
    if len(sys.argv) < 2:
        print("Usage: python create_project.py <domain> [port]")
        print("Example: python create_project.py pinescript 54322")
        print("Example: python create_project.py python 54323")
        sys.exit(1)

    domain = sys.argv[1]
    port = int(sys.argv[2]) if len(sys.argv) > 2 else 54322
    create_project(domain, port)
```

USAGE GUIDE

Step 1: Verify Template

```
cd ~/agents/templates/project_template
ls -la
```

Expected: 14 files including .gitignore

Step 2: Create New Project

```
cd ~/agents
python3 create_project.py pinescript 54322
```

Creates: ~/agents/projects/pinescript-expert/

Step 3: Navigate to Project

```
cd ~/agents/projects/pinescript-expert
```

Step 4: Add API Key

Edit .env file and add your OpenAI API key:

```
nano .env
# Change: OPENAI_API_KEY=sk-...
# To: OPENAI_API_KEY=sk-your-actual-key
```

Step 5: Start Database

```
docker-compose up -d
```

Step 6: Install Dependencies

```
pip install -r requirements.txt
```

Step 7: Initialize Database

```
python3 scripts/init_db.py
```

Step 8: Launch Interface

```
streamlit run interfaces/streamlit_app.py
```

Browser opens automatically at localhost:8501

CREATING MULTIPLE PROJECTS

```
cd ~/agents
python3 create_project.py pinescript 54320
python3 create_project.py python 54321
python3 create_project.py shopify 54322
python3 create_project.py react 54323
```

Each project is completely isolated with unique:

• Database on unique port

• Configuration files

• Memory storage

• Environment variables

VERIFICATION CHECKLIST

Template Integrity

☐ Template exists at ~/agents/templates/project_template/

☐ All 14 files present

☐ No .env files in template (only .env.example)

☐ No postgres_data folder in template

☐ No knowledge folder in template

☐ No logs folder in template

Generator Functionality

☐ create_project.py executable

☐ Creates new project folder

☐ .env created with correct port and name

☐ README.md has project name replaced

☐ project.yaml has project name replaced

☐ All subdirectories created

Project Testing

☐ docker-compose up -d succeeds

☐ pip install succeeds

☐ python scripts/init_db.py succeeds

☐ streamlit run launches browser

☐ UI loads with correct project name

☐ No async errors in console

☐ Chat functionality works

COMMON ISSUES

Issue: python command not found

Solution: Use python3 instead of python on Mac

```
python3 create_project.py domain port
```

Issue: Docker not starting

Solution: Ensure Docker Desktop is running

```
docker ps
# Should show running containers
```

Issue: Port already in use

Solution: Use different port for each project

```
docker-compose down
# Edit .env and change PORT
docker-compose up -d
```

Issue: Module not found

Solution: Install dependencies in project directory

```
cd ~/agents/projects/your-project-expert
pip install -r requirements.txt
```

ARCHITECTURE ADVANTAGES

Immutability

Template never modified. All changes happen in generated projects.

Isolation

Projects are truly independent. No shared state, databases, or configuration.

Scalability

Add unlimited domain experts. Each scales independently.

Clarity

Layer boundaries are crystal clear. No mixed responsibilities.

Maintainability

Single source of truth (template). Updates propagate to new projects.

Security

.gitignore prevents accidental exposure of secrets or private data.

PRODUCTION STATUS

Phase 1.5: COMPLETE ✓

• Template hardened and verified

• Controller initialization fixed

• load_dotenv() placement corrected

• .gitignore added

• All 14 files verified

• Layer boundaries enforced

• Project isolation confirmed

Next Phase: Phase 2 - First Project Implementation

• Choose domain (PineScript, Python, Shopify, etc.)

• Create project from template

• Add domain-specific requirements

• Implement domain agent

• Implement domain controller

• Add domain tools

• Test and validate

NOTES

Clean Code Philosophy

All files contain zero comments. Instructions and documentation live outside code in this document and in README files.

Template Immutability

The template at ~/agents/templates/project_template/ must never be edited directly. It serves as the blueprint for all future projects.

Port Management

Each project requires a unique port. Recommended range: 54320-54399 for development.

API Key Security

Never commit .env files. Always use .env.example for templates.

Python Command

On Mac, use python3 instead of python for all commands.