

PSScript Platform Implementation

Summary

Date: January 26, 2026 **Based On:** TECH-REVIEW-2026.md **Status:** In Progress

Executive Summary

This document tracks the implementation of improvements identified in TECH-REVIEW-2026.md. Many recommended upgrades have already been completed, significantly reducing the scope of remaining work.

Already Completed (No Action Needed)

Dependency Upgrades

1. **React Query:**  Already on v5.62.12 (Target: v5.x)
 2. Frontend package.json shows `@tanstack/react-query: ^5.62.12`
 3. Recommendation: Already complete!
 4. **OpenAI SDK:**  Already on v6.15.0 (Target: v4.x)
 5. Backend package.json shows `openai: ^6.15.0`
 6. Exceeds the recommended v4 upgrade
 7. Recommendation: Already complete!
 8. **LangGraph:**  Already on v1.0.5 (Target: v1.0)
 9. AI requirements.txt shows `langgraph==1.0.5`
 10. langgraph-checkpoint==2.0.12 also installed
 11. Recommendation: Already complete!
 12. **Supporting Libraries:**  Already Installed
 13. Zod: v3.24.1 (for structured outputs)
 14. cmdk: v1.0.4 (for command palette)
 15. framer-motion: v11.15.0 (for animations)
-

Critical Priority (Week 1-2)

1. pgvector Upgrade to 0.8.0 with HNSW Indexes

Current State: - Backend: pgvector v0.1.4 - Python: pgvector v0.2.3

Target: pgvector v0.8.0

Performance Impact: - 9x faster vector search queries - 100x more relevant results - HNSW graph-based indexing

Implementation Steps: 1. Update backend/package.json: `pgvector: ^0.8.0` 2. Update src/ai/requirements.txt: `pgvector==0.8.0` 3. Create migration script (docs/migrations/pgvector-0.8.0-migration.sql) 4. Run migration on database 5. Test vector search performance

Research Sources: - [AWS: Supercharging vector search with pgvector 0.8.0](#) - [pgvector 2026 guide](#)

2. FastAPI Upgrade (0.98.0 → 0.115.x)

Current State: FastAPI v0.98.0

Security Risk: Medium (outdated dependencies)

Implementation Steps: 1. Update requirements.txt: `fastapi==0.115.0` 2. Update uvicorn to latest compatible version 3. Test all API endpoints 4. Check for breaking changes in middleware

3. Agent System Consolidation

Current State Analysis:

Active Agents (Keep): - `agent_coordinator.py` - Main orchestrator  - `multi_agent_system.py` - Multi-agent framework  - `langgraph_production.py` - LangGraph 1.0 implementation 

`enhanced_memory.py` - Memory system ✓ - `tool_integration.py` - Tool registry ✓ - `task_planning.py` - Task planner ✓ - `state_visualization.py` - State tracker ✓ - `voice_agent.py` - Voice integration ✓

Legacy Agents (Archive): - `langchain_agent.py` - Superseded by LangGraph ✗ - `autogpt_agent.py` - No longer used ✗ - `hybrid_agent.py` - Redundant ✗ - `py_g_agent.py` - Experimental ✗ - `openai_assistant_agent.py` - Replaced by direct OpenAI integration ✗ - `agent_factory.py` - No longer needed after consolidation ✗

Implementation Steps: 1. Create `src/ai/agents/_archive/` directory 2. Move legacy agents to archive 3. Update imports in `main.py` to use `langgraph_production.py` 4. Remove `agent_factory.py` references 5. Test with LangGraph-only workflow

Expected Impact: - Remove ~3,500 LOC - 2.2x faster agent execution - 30-50% token cost reduction



Medium Priority (Week 3-4)

4. Implement pgBouncer Connection Pooling

Current: Direct PostgreSQL connections

Target: pgBouncer for connection pooling

Implementation: - Add pgbouncer service to docker-compose.yml - Configure pool_mode=transaction - Update backend connection.ts to use pgbouncer port 6432 - Set max_client_conn=1000, default_pool_size=25

Benefits: - Support 1000+ concurrent clients - Reduced connection overhead - Better resource utilization

5. Structured Outputs Implementation

Current: SDK v6 supports structured outputs via Zod

Implementation: 1. Create Zod schemas for AI responses (src/backend/src/schemas/) 2. Update AI service calls to use zodResponseFormat 3. Add response validation middleware 4. Test all AI endpoints

Benefits: - Guaranteed valid JSON responses - Better TypeScript integration - Reduced parsing errors

6. AI Usage Analytics Dashboard

Implementation: 1. Create AIMetrics model (track tokens, costs, latency) 2. Add analytics middleware to AI routes 3. Create analytics API endpoints (/api/analytics/ai) 4. Build dashboard UI component

Metrics to Track: - Cost by model/user/endpoint - Token usage trends - Latency percentiles (p50, p95, p99) - Error rates - Budget alerts



Low Priority (Week 5-6)

7. Remove In-Memory LRU Cache

Current: Dual caching (in-memory + Redis)

Target: Single Redis strategy

Implementation: 1. Remove LRU cache from src/backend/src/index.ts (~150 LOC)
2. Standardize on Redis with TTL strategy 3. Implement cache middleware

Benefits: - 100MB memory savings per instance - Horizontal scaling support - Cache persistence across deployments

8. UI Component Consolidation

Current: Two component systems (ui/ and ui-enhanced/)

Status: Needs audit to determine which is actively used

Implementation: 1. Audit component usage across pages 2. Consolidate to single system (likely ui-enhanced/) 3. Update imports 4. Remove duplicate directory

Expected Impact: - ~1,200 LOC reduction - Smaller bundle size - Consistent styling

Implementation Metrics

Metric	Before	After	Improvement
React Query	v3.x	<input checked="" type="checkbox"/> v5.62.12	Complete
OpenAI SDK	v3.x	<input checked="" type="checkbox"/> v6.15.0	Complete
LangGraph	None	<input checked="" type="checkbox"/> v1.0.5	Complete
pgvector (backend)	v0.1.4	v0.8.0 (pending)	9x faster
pgvector (Python)	v0.2.3	v0.8.0 (pending)	9x faster
FastAPI	v0.98.0	v0.115.x (pending)	Security fix
Agent Files	16 files	8 files (pending)	-50% complexity
Caching Systems	2 systems	1 system (pending)	-100MB RAM



Deployment Plan

Phase 1: Database & Dependencies (Week 1)

1. Verify React Query v5 compatibility
2. Verify OpenAI SDK v6 compatibility
3. Upgrade pgvector to 0.8.0
4. Upgrade FastAPI to 0.115.x
5. Run database migrations
6. Performance testing

Phase 2: Agent Consolidation (Week 2)

1. Integrate langgraph_production.py
2. Archive legacy agents
3. Update imports and references
4. Test multi-agent workflows
5. Monitor token costs

Phase 3: Infrastructure (Week 3)

1. Add pgBouncer to Docker setup
2. Implement structured outputs
3. Add AI analytics middleware
4. Remove in-memory cache
5. Performance benchmarking

Phase 4: Final Testing (Week 4)

1. End-to-end testing
 2. Load testing
 3. Security audit
 4. Documentation updates
 5. Production deployment
-



Research Sources Consulted

React Query v5

- [TanStack Query v5 Migration Guide](#)
- [Announcing TanStack Query v5](#)

OpenAI SDK v4+

- [OpenAI Node SDK v4 Migration](#)
- [Structured Outputs Guide](#)

pgvector 0.8.0

- [AWS: pgvector 0.8.0 Performance](#)
- [HNSW Indexes with Postgres](#)

LangGraph 1.0

- [LangGraph Memory Management](#)
 - [PostgreSQL Checkpointer](#)
 - [Mastering LangGraph Checkpointing 2025](#)
-

Success Criteria

- [] pgvector upgraded to 0.8.0 with HNSW indexes
 - [] Vector search queries 9x faster
 - [] FastAPI upgraded to 0.115.x
 - [] Legacy agents archived (6 files removed)
 - [] LangGraph production workflow active
 - [] Token costs reduced by 30-50%
 - [] Structured outputs implemented
 - [] AI analytics dashboard deployed
 - [] pgBouncer connection pooling active
 - [] Single Redis caching strategy
 - [] All tests passing
 - [] Performance benchmarks met
-

Next Steps: 1. Begin pgvector upgrade 2. Create migration SQL script 3. Update package dependencies 4. Test vector search performance 5. Proceed with agent consolidation

Document Version: 1.0 **Last Updated:** January 26, 2026 **Prepared By:** Claude Code