UnrealAgent2 Implementation Plan

Current Foundation

We have a solid foundation with well-defined interfaces and types:

1. **Core Interfaces**
    - `PropertyStorage`: Core storage operations interface
    - `BasePropertyStorage`: Abstract base class for implementations
    - `PropertyStorageService`: Service layer implementation
2. **Data Models**
    - `PropertyData`: Core property data structure
    - `NFTMetadata`: NFT-specific information
    - `MarketStatus`: Real-time market data
    - `SearchOptions` & `SearchResult`: Search functionality
    - `FilterGroup` & `MetadataFilter`: Filtering system

Implementation Strategy

Phase 1: Test Infrastructure (Week 1)

1. **Memory Integration Tests**

```
describe('Memory Integration', () => {
    // Test existing memory manager integration
    it('should store and retrieve knowledge');
    it('should handle vector embeddings');
    it('should perform similarity search');
});
```

2. **Storage Interface Tests**

```
describe('PropertyStorage Interface', () => {
    // Test against BasePropertyStorage implementation
    it('should implement all required methods');
    it('should handle property validation');
    it('should manage vector operations');
});
```

3. **Service Layer Tests**

```
describe('PropertyStorageService', () => {
    // Test service functionality
    it('should initialize with storage backend');
    it('should proxy storage operations');
    it('should handle errors properly');
});
```

Phase 2: UnrealAgent2 Integration (Week 2)

1. **Core Agent Tests**

```
describe('UnrealAgent2', () => {
    // Test agent initialization
    it('should initialize with proper configuration');
    it('should connect to storage service');
    it('should handle embedding generation');
});
```

2. **Query Processing Tests**

```
describe('Query Processing', () => {
    // Test natural language handling
    it('should parse property queries');
    it('should extract search parameters');
    it('should generate embeddings');
});
```

3. **Market Integration Tests**

```
describe('Market Integration', () => {
    // Test NFT market features
    it('should fetch market status');
    it('should update property listings');
    it('should track price changes');
});
```

Implementation Steps

Week 1: Core Infrastructure

Day 1-2: Memory Integration

1. Create `memory-integration.test2.ts`
2. Test knowledge storage operations

3. Validate embedding functionality
4. Test search capabilities

Day 3-4: Storage Interface

1. Create `property-storage.test.ts`
2. Test CRUD operations
3. Test search operations
4. Test bulk operations

Day 5: Service Layer

1. Create `property-service.test.ts`
2. Test service initialization
3. Test operation proxying
4. Test error handling

Week 2: UnrealAgent2 Features

Day 1-2: Agent Core

1. Create `unreal-agent2.test.ts`
2. Test configuration management
3. Test service integration
4. Test embedding operations

Day 3-4: Query Features

1. Create `query-processing.test.ts`
2. Test natural language parsing
3. Test parameter extraction
4. Test search execution

Day 5: Market Features

1. Create `market-integration.test.ts`
2. Test NFT data fetching
3. Test market updates
4. Test price tracking

Test Structure

```
tests/
├── memory-integration.test2.ts   # Enhanced memory tests
├── property-storage.test.ts      # Storage interface tests
├── property-service.test.ts      # Service layer tests
├── unreal-agent2.test.ts        # Agent core tests
├── query-processing.test.ts      # Query handling tests
└── market-integration.test.ts    # Market feature tests
```

Success Criteria

1. **Test Coverage**
   - All interfaces fully tested
   - Edge cases covered
   - Error scenarios handled
2. **Integration Points**
   - Memory manager integration verified
   - Storage service properly tested
   - Market integration validated
3. **Performance Metrics**
   - Search response times
   - Embedding generation speed
   - Market data latency

Next Steps

1. Begin with `memory-integration.test2.ts`
   - Build on existing test patterns
   - Add new test cases
   - Improve error coverage
2. Proceed with storage interface tests
   - Validate interface compliance
   - Test edge cases
   - Verify error handling
3. Move to service layer tests
   - Test initialization
   - Verify operations
   - Check error propagation