Unreal Agent 2 Test Implementation Plan

Overview

Unreal Agent 2 represents our next-generation property data management system, focusing on enhanced vector search capabilities and improved data organization. This document outlines our test-driven development approach.

Core Components to Test

1. Property Data Management

- Property storage and retrieval
- Vector embedding generation
- Metadata management
- Data versioning

2. Vector Search Engine

- Semantic similarity search
- Multi-modal property matching
- Context-aware ranking
- Hybrid search (combining vector and traditional search)

3. Query Processing

- Natural language query parsing
- Query vector generation
- Query context management
- Result filtering and ranking

Test Implementation Strategy

Phase 1: Basic Infrastructure Tests

```
describe('UnrealAgent2 Infrastructure', () => {
    describe('Runtime Setup', () => {
        it('should initialize with correct embedding model');
        it('should connect to vector database');
        it('should handle configuration changes');
    });

    describe('Data Storage', () => {
        it('should store property with vector embeddings');
        it('should maintain property metadata');
        it('should handle data versioning');
    });
});
```

Phase 2: Vector Search Core

```
describe('UnrealAgent2 Vector Search', () => {
    describe('Embedding Generation', () => {
        it('should generate consistent embeddings');
        it('should handle multi-modal inputs');
        it('should normalize vectors correctly');
    });

    describe('Similarity Search', () => {
        it('should find semantically similar properties');
        it('should rank by relevance score');
        it('should handle hybrid search queries');
    });
});
```

Phase 3: Query Processing

```
describe('UnrealAgent2 Query Processing', () => {
    describe('Natural Language Processing', () => {
        it('should extract key property attributes');
        it('should understand location context');
        it('should handle price ranges');
    });

    describe('Query Transformation', () => {
        it('should convert queries to vector space');
        it('should maintain query context');
        it('should handle multi-part queries');
    });
});
```

Detailed Test Cases

1. Property Storage Tests

```
interface TestProperty {
    id: string;
    name: string;
    location: {
        address: string;
        coordinates: [number, number];
    };
    features: string[];
    description: string;
    metadata: {
```

```
        lastUpdated: Date;
        version: number;
    };
}

describe('Property Storage', () => {
    let agent: UnrealAgent2;
    let testProperty: TestProperty;

    beforeEach(async () => {
        testProperty = {
            id: 'test-1',
            name: 'Luxury Oceanfront Villa',
            location: {
                address: '123 Ocean Drive, Miami Beach',
                coordinates: [25.7617, -80.1918]
            },
            features: ['oceanfront', 'luxury', 'villa'],
            description: 'Stunning oceanfront property with...',
            metadata: {
                lastUpdated: new Date(),
                version: 1
            }
        };
    });

    it('should store property with vector embedding', async () => {
        const result = await agent.storeProperty(testProperty);
        expect(result.embedding).toBeDefined();
        expect(result.embedding.length).toBe(1536); // OpenAI embedding
size
    });

    it('should retrieve property with context', async () => {
        const context = {
            userLocation: [25.7617, -80.1918],
            preferences: ['luxury', 'oceanfront']
        };
        const result = await agent.getProperty(testProperty.id, context);
        expect(result.relevanceScore).toBeGreaterThan(0.8);
    });
});
```

2. Vector Search Tests

```
describe('Vector Search', () => {
    describe('Semantic Search', () => {
        it('should find properties by description', async () => {
            const query = 'modern beachfront property with ocean views';
            const results = await agent.semanticSearch(query);
            expect(results[0].similarity).toBeGreaterThan(0.7);
```

```
        });

        it('should handle location-aware search', async () => {
            const query = 'properties near the beach';
            const context = {
                location: 'Miami Beach',
                radius: '5km'
            };
            const results = await agent.contextualSearch(query, context);
            expect(results).toSatisfyAll(result =>
                result.location.distance <= 5000
            );
        });
    });

    describe('Hybrid Search', () => {
        it('should combine vector and metadata search', async () => {
            const query = {
                description: 'luxury waterfront property',
                filters: {
                    priceRange: [1000000, 5000000],
                    location: 'Miami Beach'
                }
            };
            const results = await agent.hybridSearch(query);
            expect(results).toMatchSearchCriteria(query);
        });
    });
});
```

3. Query Processing Tests

```
describe('Query Processing', () => {
    describe('Natural Language Understanding', () => {
        it('should extract property attributes', async () => {
            const query = 'find me a modern 3-bedroom house near the beach
under 2M';
            const parsed = await agent.parseQuery(query);
            expect(parsed).toEqual({
                propertyType: 'house',
                bedrooms: 3,
                style: 'modern',
                location: 'near beach',
                maxPrice: 2000000
            });
        });

        it('should handle complex queries', async () => {
            const query = 'luxury condos in Miami Beach with ocean views
and a pool, between 1-3M';
            const parsed = await agent.parseQuery(query);
```

```
            expect(parsed).toMatchObject({
                propertyType: 'condo',
                features: ['ocean views', 'pool'],
                location: 'Miami Beach',
                priceRange: [1000000, 3000000],
                style: 'luxury'
            });
        });
    });
});
```

Implementation Phases

Phase 1: Foundation (Week 1)

1. Set up test infrastructure
2. Implement basic property storage tests
3. Add vector embedding generation tests
4. Create test data generators

Phase 2: Core Features (Week 2)

1. Implement vector search tests
2. Add similarity scoring tests
3. Create hybrid search tests
4. Add basic query processing tests

Phase 3: Advanced Features (Week 3)

1. Implement context-aware search tests
2. Add multi-modal search tests
3. Create complex query processing tests
4. Add performance benchmarks

Phase 4: Integration (Week 4)

1. Implement end-to-end tests
2. Add error handling tests
3. Create load tests
4. Add integration tests with other services

Test Data Strategy

1. Property Dataset

- Create a diverse set of test properties
- Include various property types
- Cover different price ranges
- Include multiple locations

2. Query Dataset

- Create realistic user queries
- Include edge cases
- Cover multiple search intents
- Include location-specific queries

3. Context Dataset

- Create user context scenarios
- Include location contexts
- Add preference profiles
- Include search history

Success Criteria

1. **Coverage**
    - 90%+ test coverage
    - All core features tested
    - Edge cases covered
2. **Performance**
    - Search results < 200ms
    - Embedding generation < 500ms
    - Query parsing < 100ms
3. **Quality**
    - Relevant search results
    - Accurate query parsing
    - Proper error handling

Next Steps

1. Create test data generators
2. Set up test infrastructure
3. Implement Phase 1 tests
4. Review and iterate on test cases

This test plan will evolve as we implement and learn more about the system's requirements and behavior.