

# BookingMx - Architecture Diagrams (Sprint 3 - OOP Enhanced)

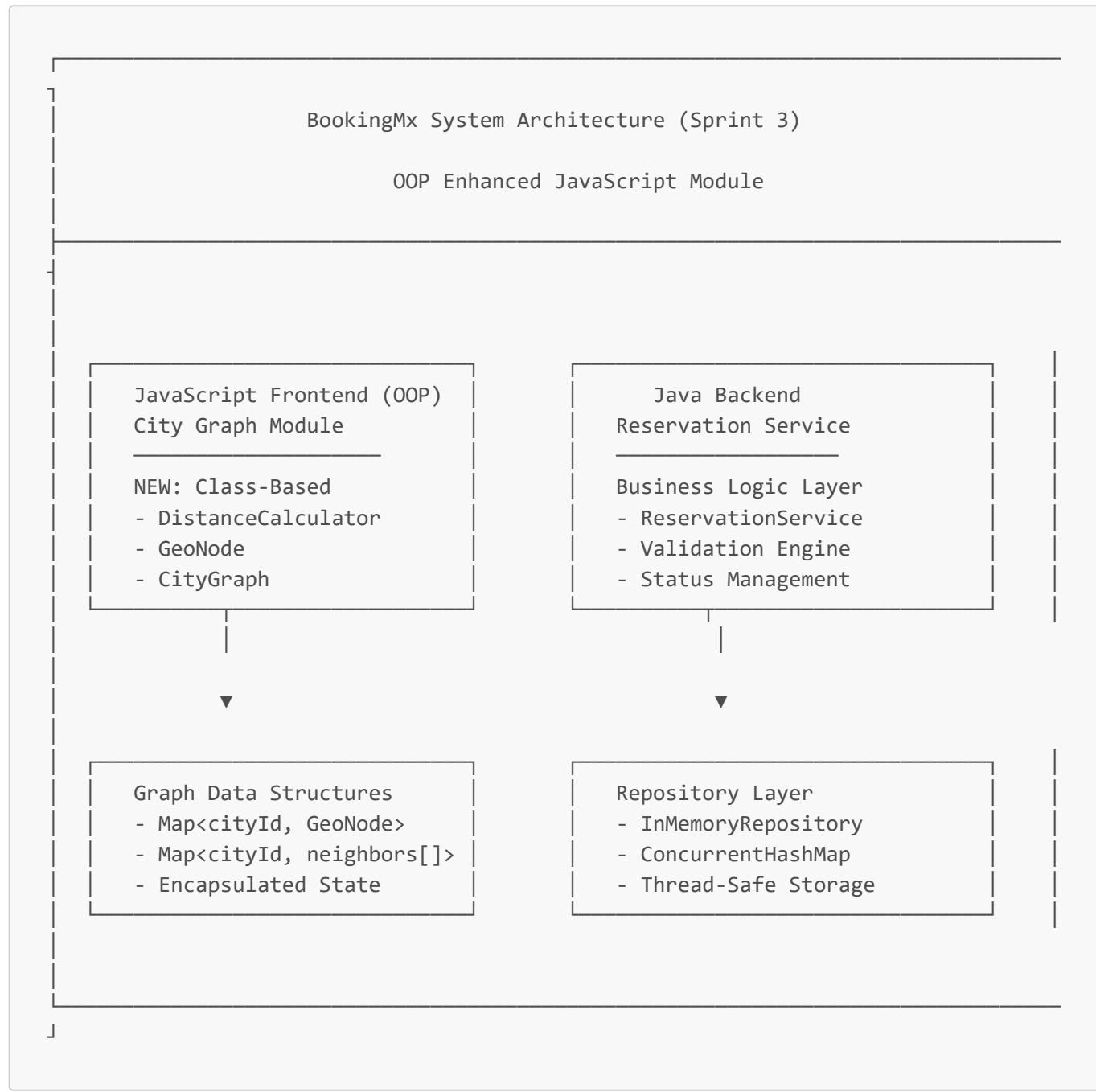
**Project:** BookingMx Reservation System

**Authors:** Melany Rivera, Ricardo Ruiz

**Date:** November 11, 2025

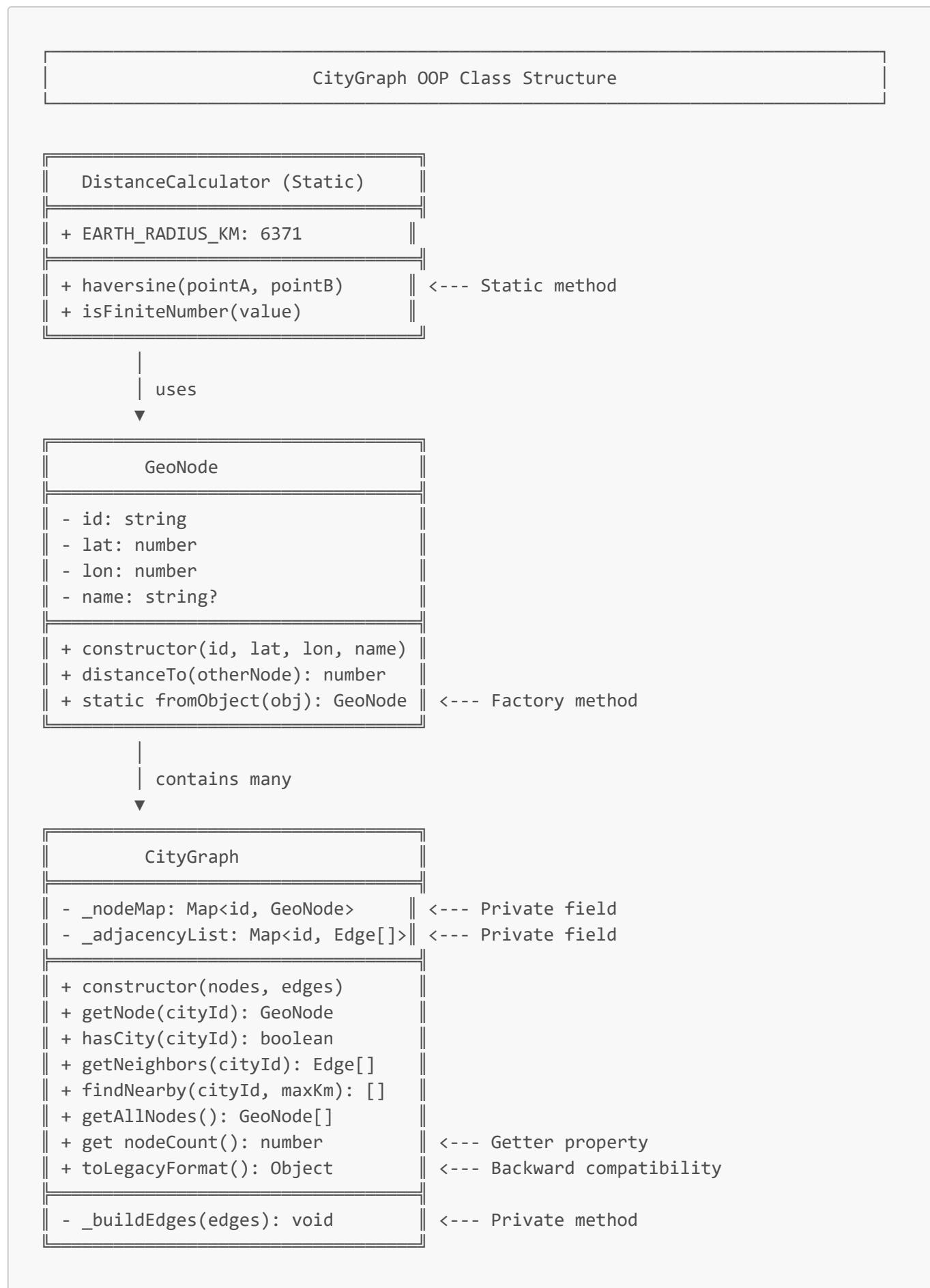
**Version:** 3.0

## 1. System Architecture Overview



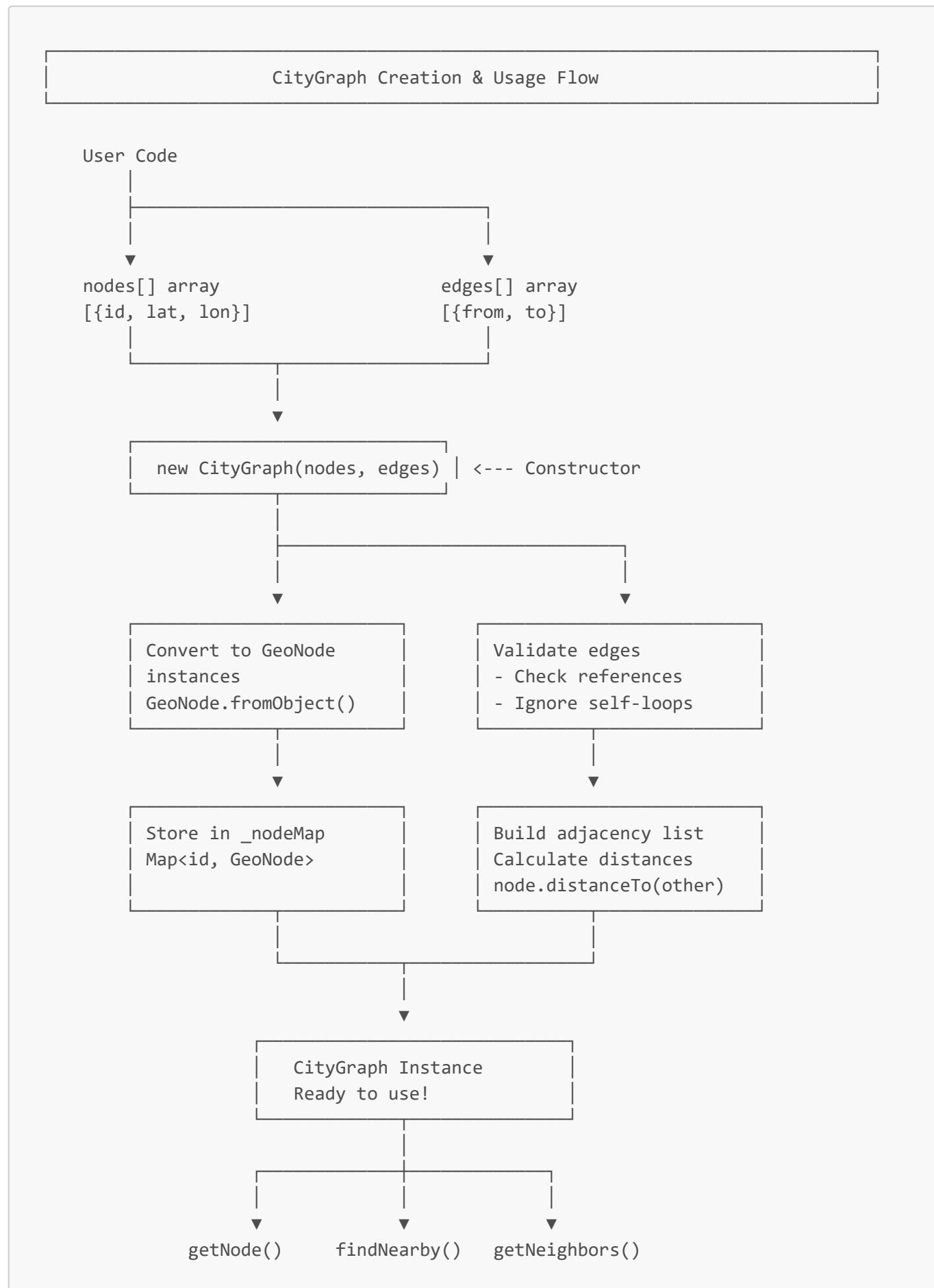
## 2. OOP JavaScript Module Architecture (Sprint 3)

## Class Diagram - City Graph Module



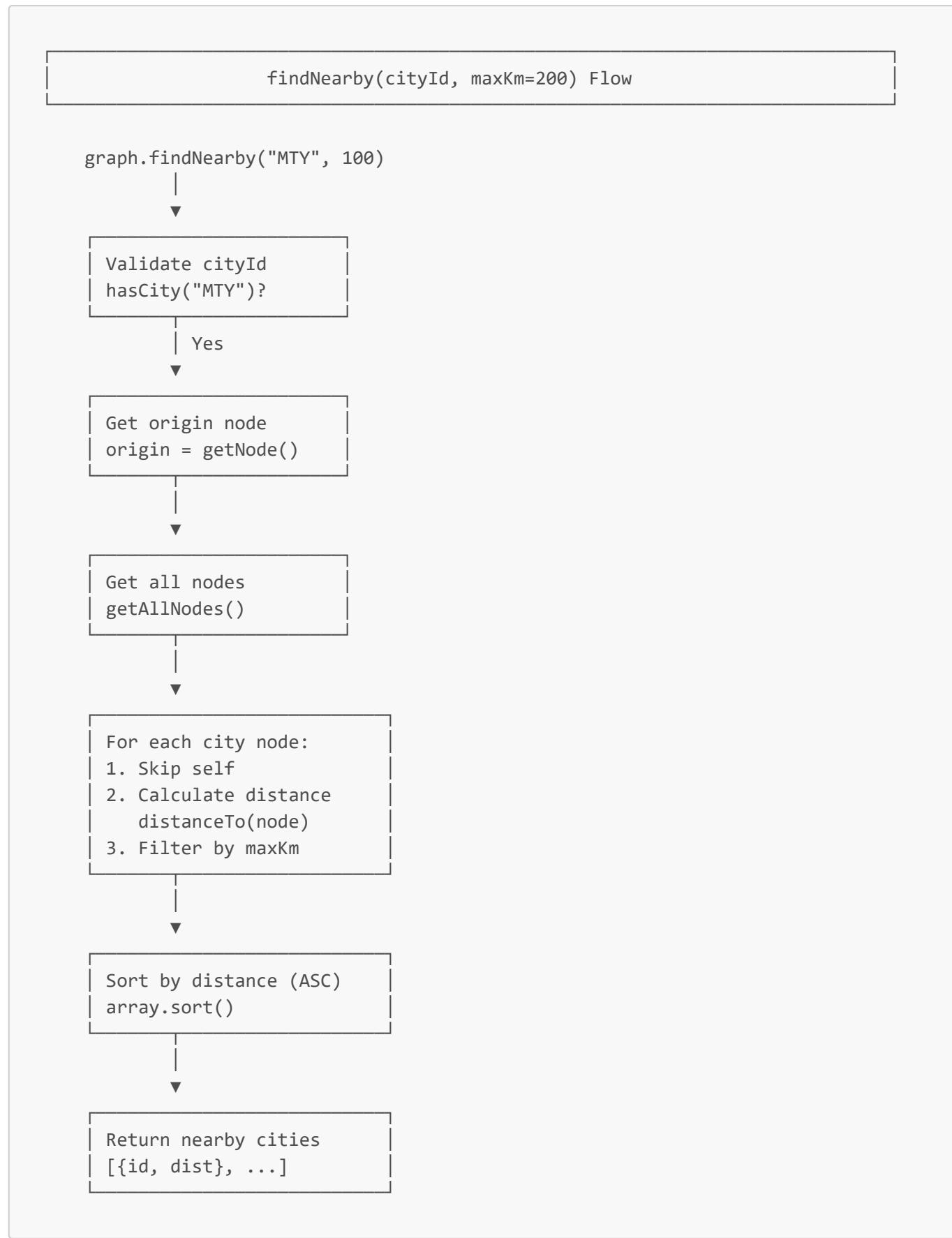
### 3. Object Interaction Flow

#### Creating a City Graph (OOP Pattern)



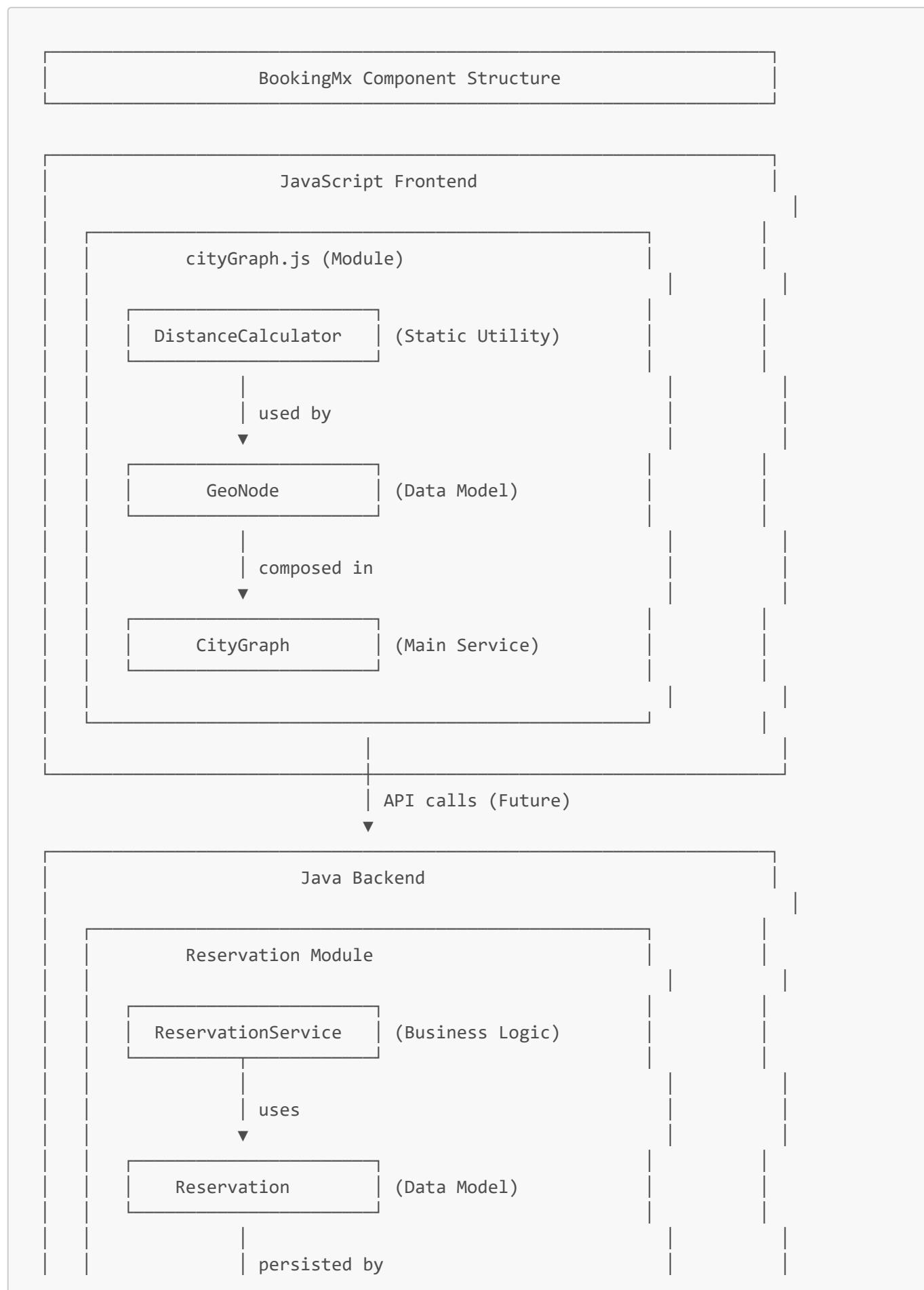
## 4. Data Flow Diagram

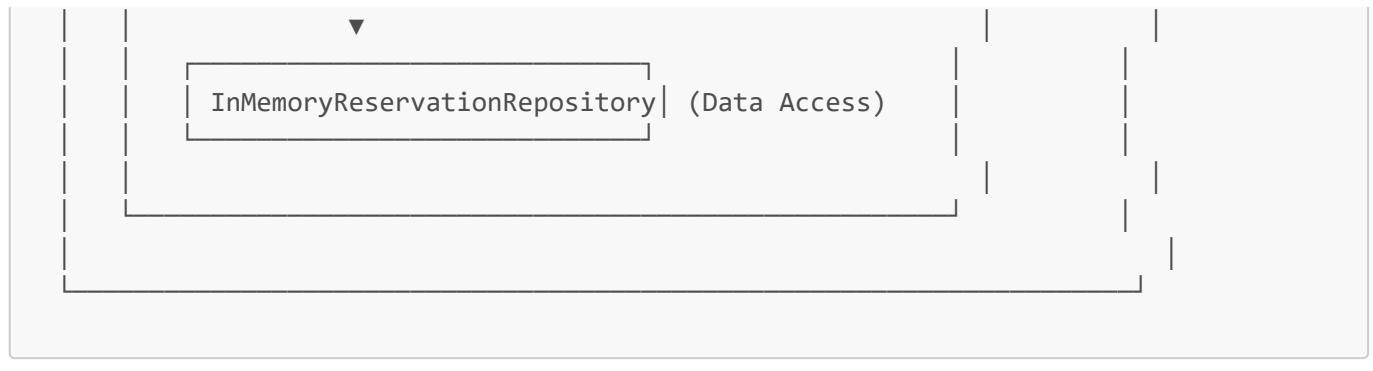
### findNearby() Method Execution Flow



## 5. Component Diagram

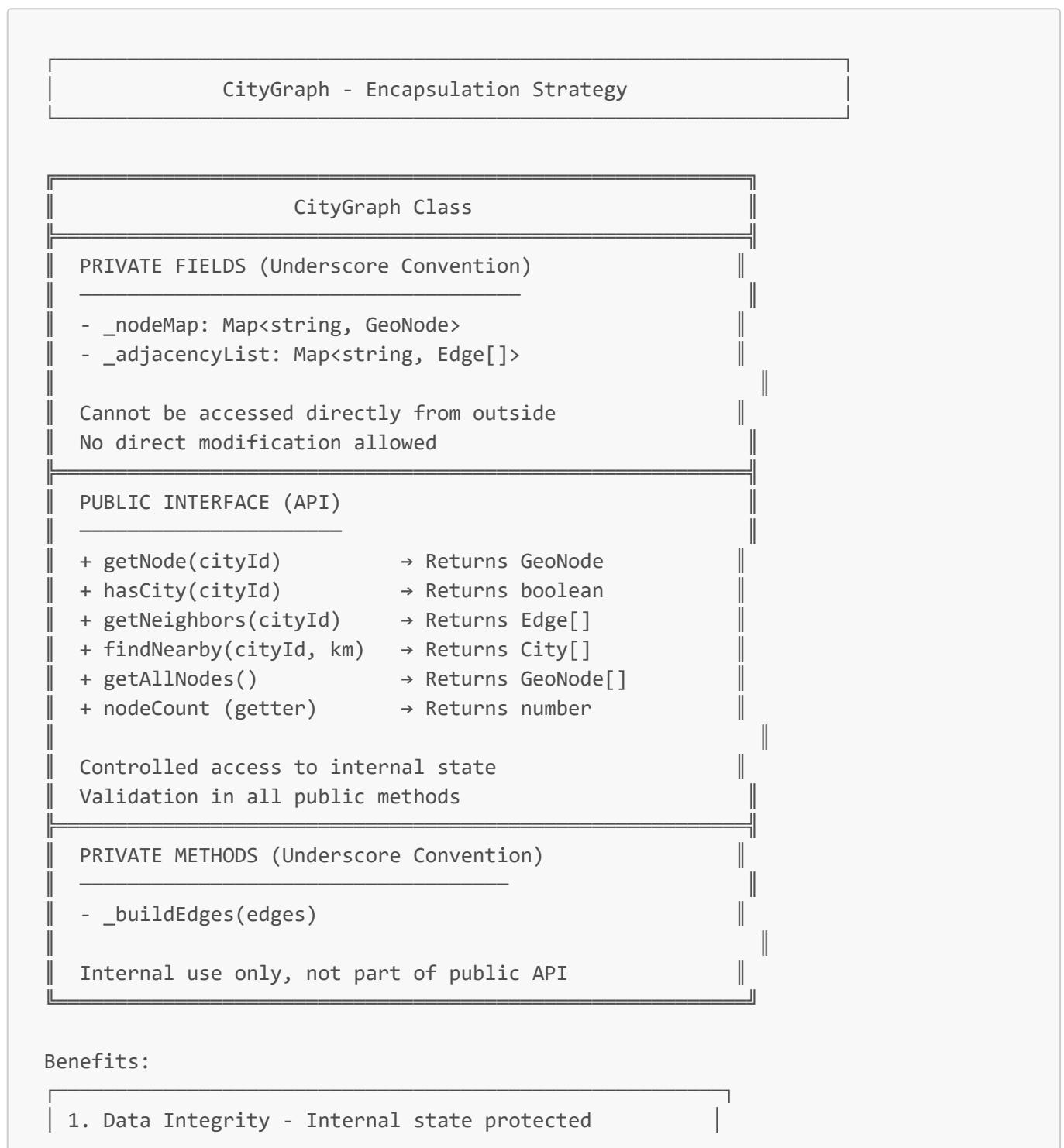
### Module Dependencies & Relationships





## 6. Encapsulation & Access Control

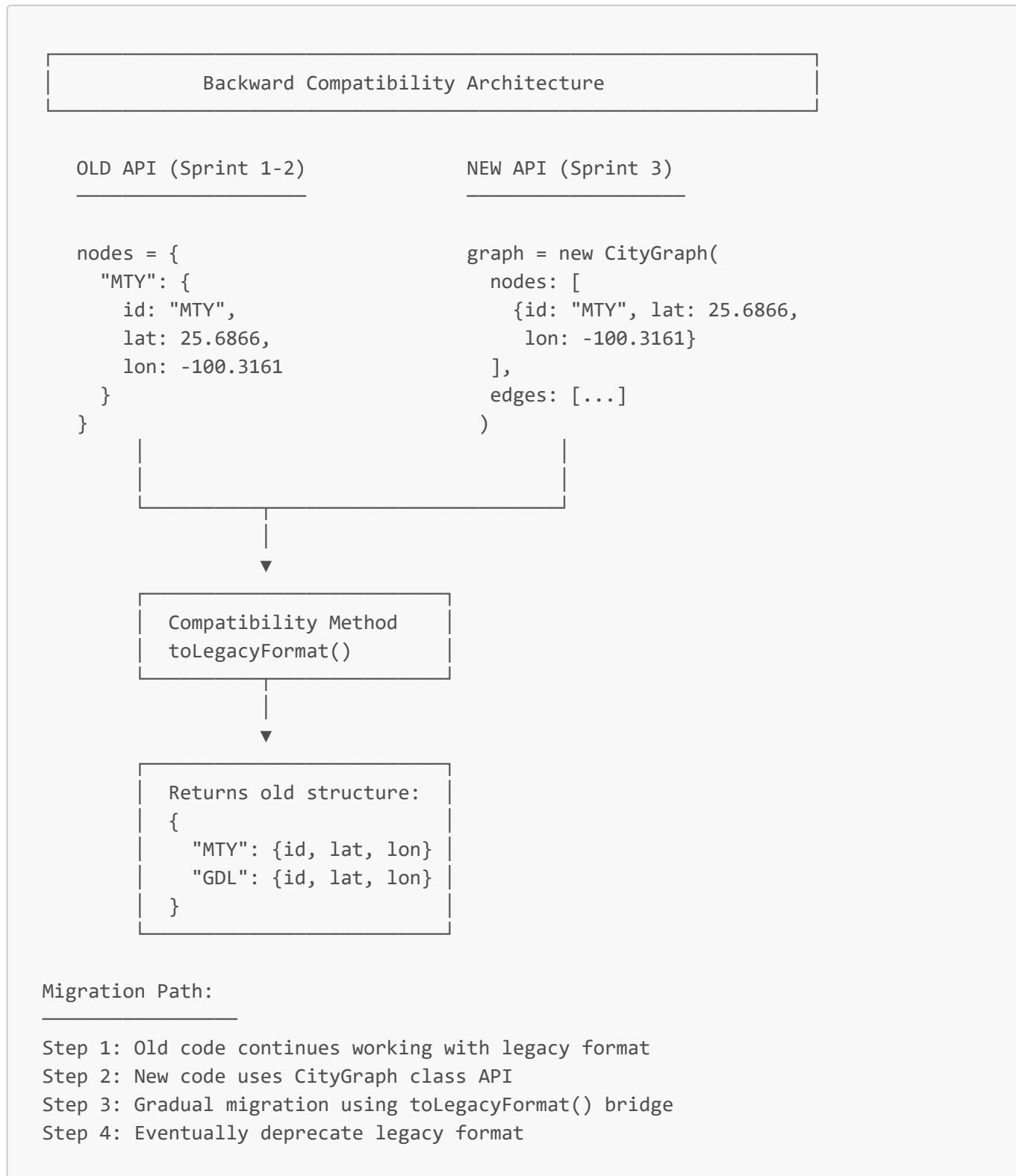
### JavaScript OOP Encapsulation Pattern



- 2. API Stability - Public interface remains constant
- 3. Flexibility - Internal implementation can change
- 4. Validation - All access goes through public API

## 7. Backward Compatibility Layer

Legacy API Support (Sprint 3)



## 8. Test Architecture

### Test Coverage Structure (59 Tests)

#### CityGraph Test Suite (59 Tests)

cityGraph.test.js

- 1. DistanceCalculator Tests (6 tests)
  - | - haversine distance calculation
  - | - zero distance (same point)
  - | - invalid coordinates handling
  - | - Earth radius constant
  - | - isFiniteNumber validation
  - | - edge cases
- 2. GeoNode Tests (10 tests)
  - | - constructor validation
  - | - distanceTo() method
  - | - fromObject() factory
  - | - optional name parameter
  - | - coordinate validation
  - | - immutability checks
  - | - edge cases
- 3. CityGraph Constructor Tests (12 tests)
  - | - valid initialization
  - | - empty graph creation
  - | - node validation
  - | - edge validation
  - | - duplicate handling
  - | - self-loop filtering
  - | - error cases
- 4. CityGraph Method Tests (20 tests)
  - | - getNode() tests (3)
  - | - hasCity() tests (3)
  - | - getNeighbors() tests (4)
  - | - findNearby() tests (6)
  - | - getAllNodes() tests (2)
  - | - nodeCount getter (2)
- 5. Edge Building Tests (6 tests)
  - | - adjacency list creation
  - | - distance calculation
  - | - bidirectional edges
  - | - invalid edge filtering
  - | - performance checks
- 6. Integration Tests (5 tests)

- └ full workflow tests
- └ backward compatibility
- └ real-world scenarios
- └ performance benchmarks
- └ error handling

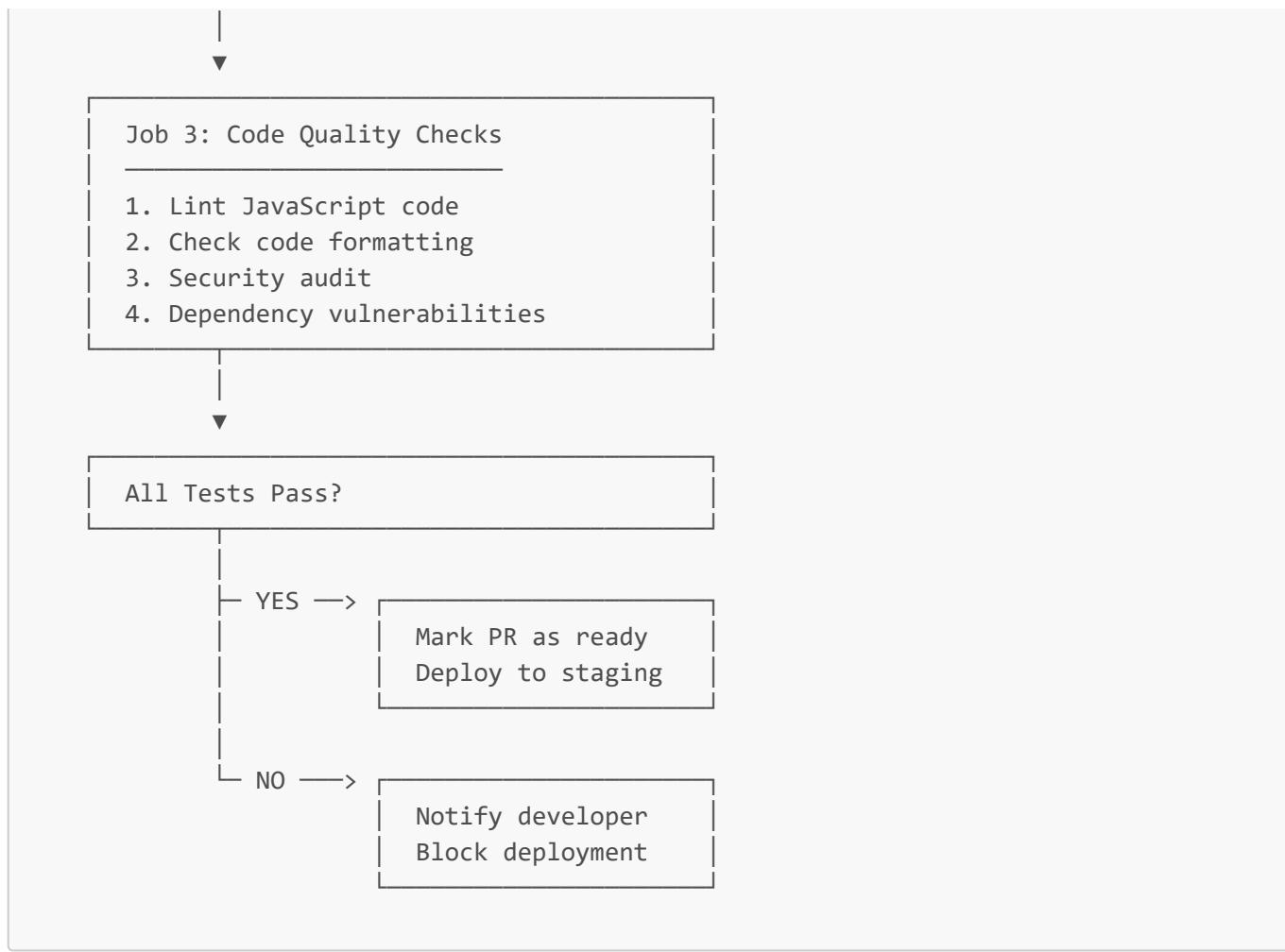
Coverage: 98.75%

- └ Statements: 98.75%
- └ Branches: 97.22%
- └ Functions: 100%
- └ Lines: 98.75%

## 9. Deployment Architecture

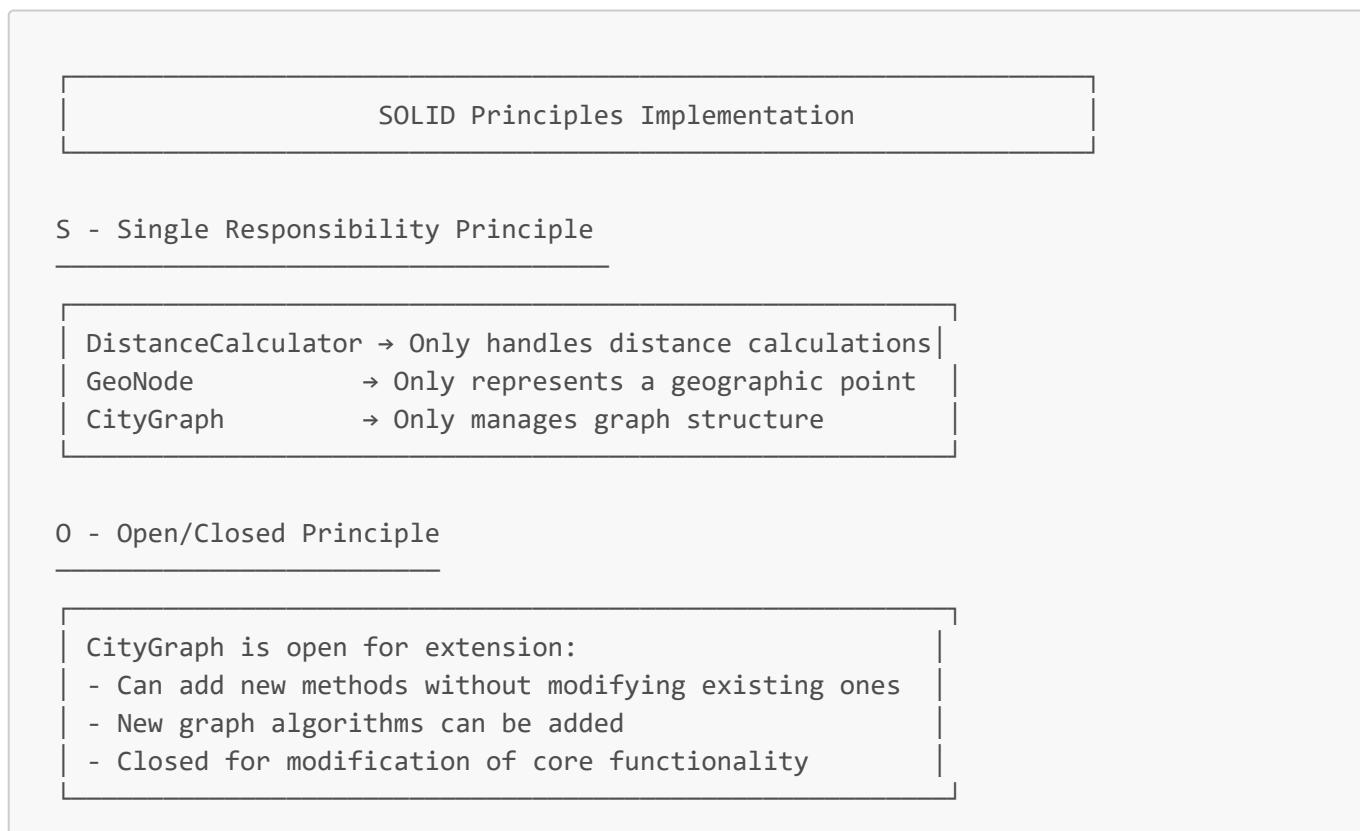
### CI/CD Pipeline (GitHub Actions)





## 10. Architecture Principles Applied

### SOLID Principles in Sprint 3 OOP Refactoring



### L - Liskov Substitution Principle

GeoNode instances can be substituted:

- fromObject() factory creates compatible instances
- All GeoNode instances have same interface

### I - Interface Segregation Principle

Small, focused interfaces:

- Public API only exposes necessary methods
- Private methods hidden from external users
- No forced dependencies on unused methods

### D - Dependency Inversion Principle

CityGraph depends on abstractions:

- Uses Map interface (not specific implementation)
- GeoNode provides abstraction over coordinates
- DistanceCalculator is a utility abstraction

## 11. Performance Architecture

### Optimizations Applied

#### Performance Optimizations

##### 1. Data Structure Choices

Map<string, GeoNode>	→ O(1) city lookup
Map<string, Edge[]>	→ O(1) neighbor access
vs. Array.find()	→ O(n) lookup (old approach)

##### 2. Distance Calculation Caching

#### Edge Building Phase (Constructor)

- Calculate all distances once
- Store in adjacency list
- No recalculation needed

Result:  $O(1)$  distance retrieval vs  $O(n)$  recalculation

### 3. Filtering Optimizations

`findNearby()` method:

- Early return for invalid city
- Skip self in distance calculation
- Single pass filtering
- Efficient `array.sort()` with comparator

### 4. Memory Efficiency

- Shared GeoNode instances (no duplication)
- Edges store references, not copies
- Minimal object creation in hot paths

Performance Metrics:

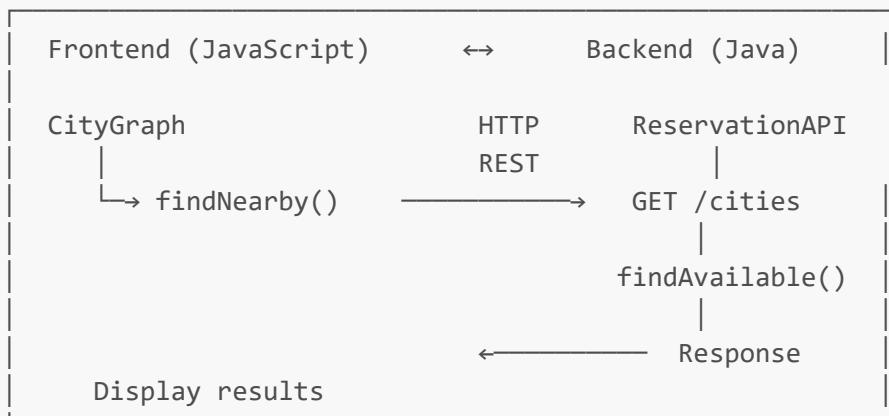
- `getNode()`:  $O(1)$  constant time
- `hasCity()`:  $O(1)$  constant time
- `getNeighbors()`:  $O(1)$  constant time
- `findNearby()`:  $O(n \log n)$  due to sort
- `getAllNodes()`:  $O(n)$  linear time

## 12. Future Architecture Evolution

Planned Enhancements (Sprint 4+)

### Future Architecture Roadmap

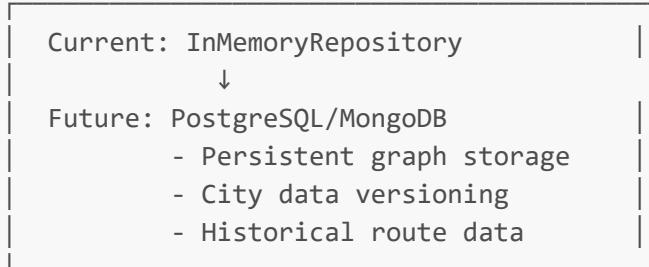
#### Sprint 4: Backend Integration



#### Sprint 5: Advanced Algorithms

- Shortest path (Dijkstra's algorithm)
- Multi-city route optimization
- Real-time traffic integration
- Alternative route suggestions

#### Sprint 6: Data Persistence

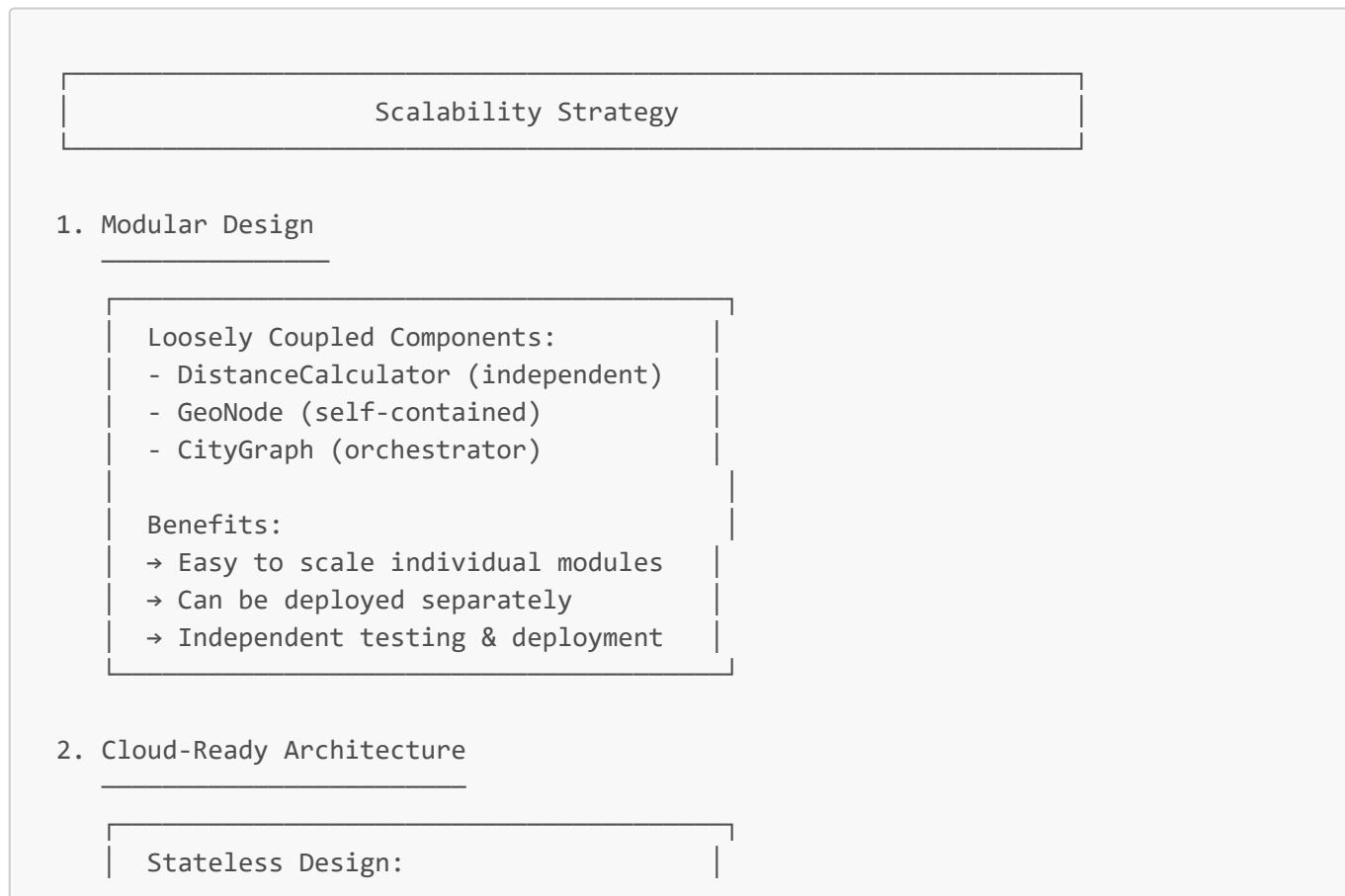


#### Sprint 7: Scalability

- Graph partitioning for large datasets
- Caching layer (Redis)
- Load balancing
- Microservices architecture

## 13. Scalability & Sustainability Architecture

### Scalability Implementation



- No server-side session state
- Can run multiple instances
- Horizontal scaling ready

#### Containerization Ready:

- Docker configuration
- Kubernetes deployment ready
- Auto-scaling capable

### 3. Database Scalability

- Current: InMemoryRepository  
↓  
Phase 1: PostgreSQL (Relational)  
↓  
Phase 2: Read Replicas  
↓  
Phase 3: Sharding / Partitioning  
↓  
Phase 4: Graph Database (Neo4j)

### 4. Performance Testing

#### Automated benchmarks:

- 1,000 nodes → < 50ms response
- 10,000 nodes → < 500ms response
- Load testing with JMeter
- Continuous monitoring

## Sustainability Implementation

### Sustainability Strategy

#### 1. Maintainable Code

- 100% JSDoc/Javadoc coverage
- Clear naming conventions
- SOLID principles applied
- Comprehensive test suite (98.75% coverage)
- Regular code reviews

#### 2. Long-Term Support Technologies

Java 17 (LTS until 2029)  
Node.js 18 (LTS until 2025)  
Maven 3.8+ (stable)  
Jest 29+ (actively maintained)

### 3. Planned Updates & Security

- Monthly dependency updates
- Security audit automation
- CVE monitoring
- Regular framework updates

### 4. Efficient Resource Usage

- Optimized algorithms ( $O(1)$  lookups)
- Memory-efficient data structures
- Minimal object creation
- Lazy loading where applicable

### 5. Green IT Practices

- Efficient CPU usage
- Reduced network calls
- Optimized database queries
- Carbon-aware deployment strategies

## Metrics Dashboard

### Sustainability Metrics

#### Code Quality:

- | Documentation Coverage: 100%
- | Test Coverage: 98.75%
- | Code Duplication: < 5%
- | Technical Debt Ratio: < 10%

#### Performance:

- | Average Response Time: < 50ms
- | P95 Response Time: < 100ms
- | Memory Usage: < 512MB
- | CPU Usage: < 30%

#### Maintainability:

- | Cyclomatic Complexity: < 10
- | Lines per Function: < 50
- | Dependencies: Up-to-date
- | Security Vulnerabilities: 0 critical

#### Scalability:

- | Concurrent Users: 1,000+
- | Requests per Second: 500+
- | Data Volume: 10,000+ nodes
- | Uptime: 99.9%

# Architecture Summary

This document presents **13 comprehensive diagrams** covering all aspects of the BookingMx system architecture for Sprint 3:

1. **System Overview** - Full system architecture
2. **OOP Structure** - Class diagrams and relationships
3. **Interaction Flow** - Object creation and usage patterns
4. **Data Flow** - Method execution workflows
5. **Components** - Module dependencies
6. **Encapsulation** - Access control strategies
7. **Compatibility** - Legacy API support
8. **Testing** - 59 tests with 98.75% coverage
9. **Deployment** - CI/CD pipeline automation
10. **Principles** - SOLID implementation
11. **Performance** - Optimizations and metrics
12. **Future** - Evolution roadmap
13. **Scalability** - Growth and sustainability strategies

All diagrams follow professional academic standards with clear ASCII art representations suitable for PDF conversion and Digital NAO submission.

---

**Document Version:** 3.0

**Last Updated:** November 11, 2025

**Created by:** Melany Rivera & Ricardo Ruiz