# AfricanMarket Testing Guide

This document provides comprehensive information about testing the AfricanMarket application, including test strategies, frameworks, and best practices.

## 🧪 Testing Strategy

### Testing Philosophy

The AfricanMarket application follows a comprehensive testing strategy that includes:
- **Unit Tests**: Test individual components and functions
- **Integration Tests**: Test component interactions
- **Performance Tests**: Ensure system performance under load
- **End-to-End Tests**: Test complete user workflows
- **Security Tests**: Verify security measures

### Test Pyramid

```
     /\
    /  \
   /E2E \     <- End-to-End Tests (Few)
  /______\
 /        \
/ Integration\ <- Integration Tests (Some)
/____________\
/            \
/  Unit Tests   \ <- Unit Tests (Many)
/________________\
```

## 🛠 Testing Framework

### Core Testing Libraries

- **Jest**: JavaScript testing framework
- **React Testing Library**: React component testing
- **Supertest**: HTTP assertion library
- **MSW**: Mock Service Worker for API mocking

### Test Structure

```
__tests__/
├── lib/                 # Service layer tests
│   ├── push-notification-service.test.ts
│   ├── websocket-service.test.ts
│   └── cache.test.ts
├── hooks/              # React hooks tests
│   ├── use-socket.test.tsx
│   └── use-push-notifications.test.tsx
├── api/                # API endpoint tests
│   └── push-notifications/
├── integration/        # Integration tests
│   └── shared-trip.test.tsx
├── performance/        # Performance tests
│   ├── websocket-performance.test.ts
│   └── push-notification-performance.test.ts
└── utils/              # Test utilities
    └── test-utils.tsx
```

## 🚀 Running Tests

### Test Runner Script

The application includes a comprehensive test runner:

```
# Run all tests
node scripts/test-runner.js

# Run specific test types
node scripts/test-runner.js --unit
node scripts/test-runner.js --integration
node scripts/test-runner.js --performance
node scripts/test-runner.js --coverage
```

### Individual Test Commands

```
# Unit tests
npx jest --testPathPattern=__tests__ --testPathIgnorePatterns=integration,performance

# Integration tests
npx jest --testPathPattern=integration

# Performance tests
npx jest --testPathPattern=performance

# Watch mode
npx jest --watch

# Coverage report
npx jest --coverage
```

## 📋 Test Configuration

### Jest Configuration

```
// jest.config.js
const nextJest = require('next/jest')

const createJestConfig = nextJest({
  dir: './',
})

const customJestConfig = {
  setupFilesAfterEnv: ['<rootDir>/jest.setup.js'],
  moduleNameMapping: {
    '^@/(.*)$': '<rootDir>/$1',
  },
  testEnvironment: 'jest-environment-jsdom',
  collectCoverageFrom: [
    'components/**/*.{js,jsx,ts,tsx}',
    'lib/**/*.{js,jsx,ts,tsx}',
    'hooks/**/*.{js,jsx,ts,tsx}',
    'app/**/*.{js,jsx,ts,tsx}',
    '!**/*.d.ts',
    '!**/node_modules/**',
  ],
  coverageReporters: ['text', 'lcov', 'html'],
  testMatch: [
    '<rootDir>/**/__tests__/**/*.{js,jsx,ts,tsx}',
    '<rootDir>/**/*.(test|spec).{js,jsx,ts,tsx}',
  ],
}

module.exports = createJestConfig(customJestConfig)
```

## Test Setup

```
// jest.setup.js
import '@testing-library/jest-dom'

// Mock Next.js router
jest.mock('next/router', () => ({
  useRouter() {
    return {
      route: '/',
      pathname: '/',
      query: {},
      asPath: '/',
      push: jest.fn(),
      pop: jest.fn(),
      reload: jest.fn(),
      back: jest.fn(),
      prefetch: jest.fn(),
      beforePopState: jest.fn(),
      events: {
        on: jest.fn(),
        off: jest.fn(),
        emit: jest.fn(),
      },
      isFallback: false,
    }
  },
}))
```

## 🔧 Unit Testing

### Testing Services

```
// Example: push-notification-service.test.ts
import { PushNotificationService } from '@/lib/push-notification-service'
import { prisma } from '@/lib/db'

jest.mock('@/lib/db')
const mockPrisma = prisma as jest.Mocked<typeof prisma>

describe('PushNotificationService', () => {
  let service: PushNotificationService

  beforeEach(() => {
    service = PushNotificationService.getInstance()
    jest.clearAllMocks()
  })

  describe('subscribeUser', () => {
    it('should create a new subscription', async () => {
      const userId = 'test-user-id'
      const subscription = {
        endpoint: 'https://test.com/push',
        keys: {
          p256dh: 'test-p256dh',
          auth: 'test-auth'
        }
      }

      mockPrisma.pushSubscription.findFirst.mockResolvedValue(null)
      mockPrisma.pushSubscription.create.mockResolvedValue({
        id: 'test-subscription-id',
        userId,
        endpoint: subscription.endpoint,
        p256dhKey: subscription.keys.p256dh,
        authKey: subscription.keys.auth,
        deviceInfo: {},
        isActive: true,
        createdAt: new Date(),
        lastUsed: new Date()
      })

      const result = await service.subscribeUser(userId, subscription)

      expect(result).toBe(true)
      expect(mockPrisma.pushSubscription.create).toHaveBeenCalledWith({
        data: {
          userId,
          endpoint: subscription.endpoint,
          p256dhKey: subscription.keys.p256dh,
          authKey: subscription.keys.auth,
          deviceInfo: {},
          isActive: true,
          lastUsed: expect.any(Date)
        }
      })
    })
  })
})
```

## Testing React Hooks

```tsx
// Example: use-push-notifications.test.tsx
import { renderHook, act } from '@testing-library/react'
import { usePushNotifications } from '@/hooks/use-push-notifications'

describe('usePushNotifications', () => {
  it('should initialize with correct default state', async () => {
    const { result } = renderHook(() => usePushNotifications())

    expect(result.current.loading).toBe(false)
    expect(result.current.error).toBe(null)
    expect(result.current.subscribed).toBe(false)
  })

  it('should subscribe to push notifications', async () => {
    const { result } = renderHook(() => usePushNotifications())

    await act(async () => {
      const success = await result.current.subscribe()
      expect(success).toBe(true)
    })

    expect(result.current.subscribed).toBe(true)
  })
})
```

## 🔗 Integration Testing

### Testing Page Components

```
// Example: shared-trip.test.tsx
import { render, screen, waitFor } from '@testing-library/react'
import SharedTripPage from '@/app/shared-trip/[token]/page'

describe('SharedTripPage Integration', () => {
  it('should render shared trip data correctly', async () => {
    const mockTripData = {
      tripShare: {
        id: 'test-share-id',
        shareToken: 'test-token',
        status: 'ACTIVE',
        ride: {
          id: 'test-ride-id',
          status: 'IN_PROGRESS',
          pickupAddress: '123 Main St',
          destinationAddress: '456 Oak Ave',
          customer: {
            id: 'customer-id',
            name: 'John Doe',
            phone: '+1234567890',
            avatar: null
          }
        }
      }
    }

    render(<SharedTripPage params={{ token: 'test-token' }} />)

    await waitFor(() => {
      expect(screen.getByText('Shared Trip')).toBeInTheDocument()
      expect(screen.getByText('John Doe has shared their ride with you')).toBeInTheDoc-
ument()
    })
  })
})
```

### Testing API Endpoints

```
// Example: subscribe.test.ts
import { NextRequest } from 'next/server'
import { POST } from '@/app/api/push-notifications/subscribe/route'

describe('/api/push-notifications/subscribe', () => {
  it('should subscribe user successfully', async () => {
    const subscriptionData = {
      userId: 'test-user-id',
      subscription: {
        endpoint: 'https://test.com/push',
        keys: {
          p256dh: 'test-p256dh',
          auth: 'test-auth'
        }
      }
    }

    const request = new NextRequest('http://localhost:3000/api/push-notifications/sub-
scribe', {
      method: 'POST',
      body: JSON.stringify(subscriptionData)
    })

    const response = await POST(request)
    const data = await response.json()

    expect(response.status).toBe(200)
    expect(data.success).toBe(true)
  })
})
```

## ⚡ Performance Testing

### WebSocket Performance

```
// Example: websocket-performance.test.ts
describe('WebSocket Performance Tests', () => {
  it('should handle multiple concurrent connections efficiently', async () => {
    const startTime = performance.now()
    const numConnections = 1000

    // Simulate multiple users connecting
    for (let i = 0; i < numConnections; i++) {
      service.joinRideRoom(`user-${i}`, `ride-${i % 10}`)
    }

    const endTime = performance.now()
    const duration = endTime - startTime

    // Should handle 1000 connections in less than 100ms
    expect(duration).toBeLessThan(100)
  })
})
```

### Load Testing

```
// Example: push-notification-performance.test.ts
describe('Push Notification Performance Tests', () => {
  it('should handle batch notification sending efficiently', async () => {
    const numNotifications = 100
    const notifications = Array.from({ length: numNotifications }, (_, i) => ({
      userId: `user-${i}`,
      type: 'test',
      template: {
        title: `Notification ${i}`,
        body: `This is notification number ${i}`,
        data: { id: i }
      }
    }))

    const startTime = performance.now()

    const result = await service.batchSendNotifications(notifications)

    const endTime = performance.now()
    const duration = endTime - startTime

    // Should process 100 notifications in reasonable time
    expect(duration).toBeLessThan(1000)
    expect(result.total).toBe(numNotifications)
  })
})
```

## 🛡️ Security Testing

### Authentication Testing

```
describe('Authentication Security', () => {
  it('should require authentication for protected routes', async () => {
    const request = new NextRequest('http://localhost:3000/api/protected')

    const response = await handler(request)

    expect(response.status).toBe(401)
  })

  it('should validate JWT tokens', async () => {
    const invalidToken = 'invalid-token'
    const request = new NextRequest('http://localhost:3000/api/protected', {
      headers: { authorization: `Bearer ${invalidToken}` }
    })

    const response = await handler(request)

    expect(response.status).toBe(401)
  })
})
```

### Input Validation Testing

```
describe('Input Validation', () => {
  it('should sanitize user input', async () => {
    const maliciousInput = '<script>alert("xss")</script>'

    const sanitized = sanitizers.xss(maliciousInput)

    expect(sanitized).not.toContain('<script>')
    expect(sanitized).toContain('&lt;script&gt;')
  })
})
```

## 📊 Coverage Reports

### Generating Coverage

```
# Generate coverage report
npx jest --coverage

# Open coverage report
open coverage/lcov-report/index.html
```

### Coverage Targets

- **Statements**: > 80%
- **Branches**: > 75%
- **Functions**: > 85%
- **Lines**: > 80%

### Coverage Configuration

```
// jest.config.js
module.exports = {
  collectCoverageFrom: [
    'lib/**/*.{js,ts}',
    'components/**/*.{js,jsx,ts,tsx}',
    'hooks/**/*.{js,ts}',
    '!**/*.d.ts',
    '!**/node_modules/**',
    '!coverage/**',
  ],
  coverageThreshold: {
    global: {
      branches: 75,
      functions: 85,
      lines: 80,
      statements: 80,
    },
  },
}
```

## 🔍 Testing Best Practices

### Test Organization

1. **Arrange**: Set up test data and mocks
2. **Act**: Execute the code being tested
3. **Assert**: Verify the expected outcome

### Naming Conventions

```
describe('ComponentName', () => {
  describe('when condition', () => {
    it('should do something', () => {
      // Test implementation
    })
  })
})
```

### Mock Management

```
// Good: Create reusable mocks
const mockUser = {
  id: 'test-user-id',
  email: 'test@example.com',
  name: 'Test User'
}

// Good: Reset mocks between tests
beforeEach(() => {
  jest.clearAllMocks()
})
```

### Test Data Management

```
// Use factory functions for test data
const createMockUser = (overrides = {}) => ({
  id: 'test-user-id',
  email: 'test@example.com',
  name: 'Test User',
  ...overrides
})
```

## 🚨 Testing Troubleshooting

### Common Issues

**Memory Issues**:

```
# Increase Node.js memory limit
export NODE_OPTIONS="--max-old-space-size=4096"
```

**Timeout Issues**:

```
// Increase timeout for slow tests
jest.setTimeout(30000)
```

**Mock Issues**:

```
// Clear mocks properly
afterEach(() => {
  jest.clearAllMocks()
  jest.restoreAllMocks()
})
```

### Debug Mode

```
# Run tests in debug mode
node --inspect-brk node_modules/.bin/jest --runInBand
```

## 📋 Continuous Integration

### GitHub Actions Example

```
name: Test Suite

on: [push, pull_request]

jobs:
  test:
    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v2

    - name: Setup Node.js
      uses: actions/setup-node@v2
      with:
        node-version: '18'
        cache: 'yarn'

    - name: Install dependencies
      run: yarn install --frozen-lockfile

    - name: Run tests
      run: node scripts/test-runner.js

    - name: Upload coverage
      uses: codecov/codecov-action@v1
      with:
        file: ./coverage/lcov.info
```

This testing guide provides comprehensive information for maintaining high-quality code through effective testing strategies. Follow these practices to ensure the reliability and maintainability of the AfricanMarket application.