Testing Guide

Overview

This document outlines the testing strategy, procedures, and guidelines for the Berean Bible Reading Plan Application.

Testing Strategy

Testing Pyramid

1. **Unit Tests** (70%)
   - Component testing
   - Utility function testing
   - API integration testing

2. **Integration Tests** (20%)
   - Component interaction testing
   - API workflow testing
   - User journey testing

3. **End-to-End Tests** (10%)
   - Critical user paths
   - Cross-browser testing
   - Performance testing

Testing Framework Setup

Dependencies

```
{
  "devDependencies": {
    "@testing-library/react": "^13.4.0",
    "@testing-library/jest-dom": "^5.16.5",
    "@testing-library/user-event": "^14.4.3",
    "jest": "^29.5.0",
    "jest-environment-jsdom": "^29.5.0",
    "cypress": "^12.17.0",
    "msw": "^1.2.2"
  }
}
```

Jest Configuration (jest.config.js)

```
const nextJest = require('next/jest')

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

const customJestConfig = {
  setupFilesAfterEnv: ['<rootDir>/jest.setup.js'],
  moduleNameMapping: {
    '^@/components/(.*)$': '<rootDir>/components/$1',
    '^@/pages/(.*)$': '<rootDir>/pages/$1',
    '^@/lib/(.*)$': '<rootDir>/lib/$1',
  },
  testEnvironment: 'jest-environment-jsdom',
  collectCoverageFrom: [
    'app/**/*.{js,jsx,ts,tsx}',
    '!app/**/*.d.ts',
    '!app/**/index.{js,jsx,ts,tsx}',
  ],
  coverageThreshold: {
    global: {
      branches: 80,
      functions: 80,
      lines: 80,
      statements: 80,
    },
  },
}

module.exports = createJestConfig(customJestConfig)
```

Jest Setup (jest.setup.js)

```
import '@testing-library/jest-dom'
import { server } from './mocks/server'

// 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().mockResolvedValue(undefined),
      beforePopState: jest.fn(),
      events: {
        on: jest.fn(),
        off: jest.fn(),
        emit: jest.fn(),
      },
    }
  },
}))

// Setup MSW
beforeAll(() => server.listen())
afterEach(() => server.resetHandlers())
afterAll(() => server.close())
```

Unit Testing

Component Testing

Example: Bible Verse Component Test

```
// __tests__/components/BibleVerse.test.tsx
import { render, screen } from '@testing-library/react'
import BibleVerse from '@/components/BibleVerse'

describe('BibleVerse Component', () => {
  const mockVerse = {
    reference: 'John 3:16',
    text: 'For God so loved the world...',
    translation: 'ESV'
  }

  it('renders verse reference correctly', () => {
    render(<BibleVerse verse={mockVerse} />)
    expect(screen.getByText('John 3:16')).toBeInTheDocument()
  })

  it('renders verse text correctly', () => {
    render(<BibleVerse verse={mockVerse} />)
    expect(screen.getByText(/For God so loved the world/)).toBeInTheDocument()
  })

  it('displays translation abbreviation', () => {
    render(<BibleVerse verse={mockVerse} />)
    expect(screen.getByText('ESV')).toBeInTheDocument()
  })

  it('handles missing verse data gracefully', () => {
    render(<BibleVerse verse={null} />)
    expect(screen.getByText(/No verse available/)).toBeInTheDocument()
  })
})
```

Utility Function Testing

Example: Bible Reference Parser Test

```
// __tests__/lib/bibleUtils.test.ts
import { parseReference, formatReference } from '@/lib/bibleUtils'

describe('Bible Utilities', () => {
  describe('parseReference', () => {
    it('parses single verse reference', () => {
      const result = parseReference('John 3:16')
      expect(result).toEqual({
        book: 'John',
        chapter: 3,
        verse: 16
      })
    })

    it('parses chapter range reference', () => {
      const result = parseReference('Genesis 1-3')
      expect(result).toEqual({
        book: 'Genesis',
        startChapter: 1,
        endChapter: 3
      })
    })

    it('handles invalid references', () => {
      expect(() => parseReference('Invalid Reference')).toThrow()
    })
  })

  describe('formatReference', () => {
    it('formats single verse', () => {
      const result = formatReference({ book: 'John', chapter: 3, verse: 16 })
      expect(result).toBe('John 3:16')
    })

    it('formats chapter range', () => {
      const result = formatReference({
        book: 'Genesis',
        startChapter: 1,
        endChapter: 3
      })
      expect(result).toBe('Genesis 1-3')
    })
  })
})
```

API Testing with MSW

Mock Server Setup

```
// mocks/handlers.js
import { rest } from 'msw'

export const handlers = [
  rest.get('https://api.esv.org/v3/passage/text/', (req, res, ctx) => {
    const query = req.url.searchParams.get('q')

    return res(
      ctx.json({
        query,
        passages: [`Mock passage for ${query}`],
        canonical: query
      })
    )
  }),

  rest.get('/api/progress', (req, res, ctx) => {
    return res(
      ctx.json({
        currentDay: 1,
        completedDays: [1],
        totalDays: 365,
        streak: 1
      })
    )
  })
]
```

```
// mocks/server.js
import { setupServer } from 'msw/node'
import { handlers } from './handlers'

export const server = setupServer(...handlers)
```

Integration Testing

User Interaction Testing

```
// __tests__/integration/ReadingPlan.test.tsx
import { render, screen, waitFor } from '@testing-library/react'
import userEvent from '@testing-library/user-event'
import ReadingPlan from '@/components/ReadingPlan'

describe('Reading Plan Integration', () => {
  it('allows user to navigate between days', async () => {
    const user = userEvent.setup()
    render(<ReadingPlan />)

    // Check initial state
    expect(screen.getByText('Day 1')).toBeInTheDocument()

    // Navigate to next day
    await user.click(screen.getByRole('button', { name: /next day/i }))

    await waitFor(() => {
      expect(screen.getByText('Day 2')).toBeInTheDocument()
    })
  })

  it('marks day as complete when user finishes reading', async () => {
    const user = userEvent.setup()
    render(<ReadingPlan />)

    // Mark as complete
    await user.click(screen.getByRole('button', { name: /mark complete/i }))

    await waitFor(() => {
      expect(screen.getByText(/completed/i)).toBeInTheDocument()
    })
  })
})
```

API Integration Testing

```
// __tests__/integration/api.test.ts
import { getBiblePassage } from '@/lib/api/bible'

describe('Bible API Integration', () => {
  it('fetches passage successfully', async () => {
    const passage = await getBiblePassage('John 3:16', 'ESV')

    expect(passage).toHaveProperty('text')
    expect(passage).toHaveProperty('reference')
    expect(passage.reference).toBe('John 3:16')
  })

  it('handles API errors gracefully', async () => {
    // Mock API failure
    const consoleSpy = jest.spyOn(console, 'error').mockImplementation()

    const passage = await getBiblePassage('Invalid Reference', 'ESV')

    expect(passage).toBeNull()
    expect(consoleSpy).toHaveBeenCalled()

    consoleSpy.mockRestore()
  })
})
```

End-to-End Testing

Cypress Configuration

```
// cypress.config.js
import { defineConfig } from 'cypress'

export default defineConfig({
  e2e: {
    baseUrl: 'http://localhost:3000',
    setupNodeEvents(on, config) {
      // implement node event listeners here
    },
    env: {
      ESV_API_KEY: 'test-api-key'
    }
  },
  component: {
    devServer: {
      framework: 'next',
      bundler: 'webpack',
    },
  },
})
```

E2E Test Examples

```
// cypress/e2e/reading-plan.cy.js
describe('Reading Plan E2E', () => {
  beforeEach(() => {
    cy.visit('/')
  })

  it('completes a full reading session', () => {
    // Start reading plan
    cy.get('[data-testid="start-reading"]').click()

    // Navigate through passages
    cy.get('[data-testid="bible-passage"]').should('be.visible')
    cy.get('[data-testid="next-passage"]').click()

    // Mark as complete
    cy.get('[data-testid="mark-complete"]').click()

    // Verify completion
    cy.get('[data-testid="completion-message"]').should('contain', 'Completed')
  })

  it('persists progress across sessions', () => {
    // Complete day 1
    cy.get('[data-testid="start-reading"]').click()
    cy.get('[data-testid="mark-complete"]').click()

    // Reload page
    cy.reload()

    // Check progress is maintained
    cy.get('[data-testid="progress-indicator"]').should('contain', '1 day')
  })
})
```

Cross-Browser Testing

```
// cypress/e2e/cross-browser.cy.js
describe('Cross-Browser Compatibility', () => {
  const browsers = ['chrome', 'firefox', 'edge']

  browsers.forEach(browser => {
    it(`works correctly in ${browser}`, () => {
      cy.visit('/')
      cy.get('[data-testid="app-title"]').should('be.visible')
      cy.get('[data-testid="reading-plan"]').should('be.visible')
    })
  })
})
```

Performance Testing

Lighthouse CI Configuration

```
// lighthouserc.js
module.exports = {
  ci: {
    collect: {
      url: ['http://localhost:3000'],
      startServerCommand: 'npm start',
    },
    assert: {
      assertions: {
        'categories:performance': ['warn', { minScore: 0.9 }],
        'categories:accessibility': ['error', { minScore: 0.9 }],
        'categories:best-practices': ['warn', { minScore: 0.9 }],
        'categories:seo': ['warn', { minScore: 0.9 }],
      },
    },
    upload: {
      target: 'temporary-public-storage',
    },
  },
}
```

Performance Test Example

```
// __tests__/performance/loading.test.ts
import { performance } from 'perf_hooks'

describe('Performance Tests', () => {
  it('loads Bible passage within acceptable time', async () => {
    const start = performance.now()

    // Simulate API call
    await getBiblePassage('John 3:16', 'ESV')

    const end = performance.now()
    const loadTime = end - start

    expect(loadTime).toBeLessThan(2000) // 2 seconds max
  })
})
```

Test Scripts

Package.json Scripts

```
{
  "scripts": {
    "test": "jest",
    "test:watch": "jest --watch",
    "test:coverage": "jest --coverage",
    "test:e2e": "cypress run",
    "test:e2e:open": "cypress open",
    "test:lighthouse": "lhci autorun",
    "test:all": "npm run test && npm run test:e2e && npm run test:lighthouse"
  }
}
```

Continuous Integration

GitHub Actions Test Workflow

```
# .github/workflows/test.yml
name: Tests

on: [push, pull_request]

jobs:
  test:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v3
      - uses: actions/setup-node@v3
        with:
          node-version: '18'
          cache: 'npm'

      - run: npm ci
      - run: npm run test:coverage
      - run: npm run build
      - run: npm start &
      - run: npm run test:e2e
      - run: npm run test:lighthouse

      - name: Upload coverage
        uses: codecov/codecov-action@v3
```

Testing Best Practices

General Guidelines

1. **Test Naming**: Use descriptive test names that explain the expected behavior
2. **Test Structure**: Follow Arrange-Act-Assert pattern
3. **Test Isolation**: Each test should be independent and not rely on others
4. **Mock External Dependencies**: Use MSW for API mocking
5. **Test User Behavior**: Focus on testing what users actually do

Component Testing Guidelines

1. **Test Props**: Verify components handle different prop combinations
2. **Test Events**: Ensure event handlers are called correctly
3. **Test Accessibility**: Include accessibility testing in component tests
4. **Test Error States**: Verify graceful error handling

API Testing Guidelines

1. **Test Success Cases**: Verify successful API responses
2. **Test Error Cases**: Test network failures and API errors
3. **Test Rate Limiting**: Verify rate limiting behavior
4. **Test Caching**: Ensure caching works correctly

Coverage Requirements

Minimum Coverage Thresholds

- **Statements**: 80%
- **Branches**: 80%
- **Functions**: 80%
- **Lines**: 80%

Coverage Exclusions

- Configuration files
- Test files themselves
- Build and deployment scripts
- Third-party integrations (covered by integration tests)

For questions about testing procedures or to report testing issues, contact the development team or create an issue in the repository.