

Testing Setup Instructions

Completed Setup

The testing infrastructure has been successfully configured with:

- **Vitest** - Fast unit test runner
- **@testing-library/react** - React component testing utilities
- **@testing-library/jest-dom** - DOM matchers
- **jsdom** - DOM environment for tests
- **happy-dom** - Alternative lightweight DOM

Files Created

Configuration

- `vitest.config.ts` - Vitest configuration
- `vitest.setup.ts` - Test setup and mocks

Test Files

- `__tests__/components/button.test.tsx`
- `__tests__/components/kpi-card.test.tsx`
- `__tests__/lib/sanitize.test.ts`
- `__tests__/lib/utils.test.ts`

Manual Setup Required

Add Test Scripts to `package.json`

Since `package.json` cannot be modified programmatically, please manually add these scripts:

```
"scripts": {
  "dev": "next dev",
  "build": "next build",
  "start": "next start",
  "lint": "next lint",
  "test": "vitest",
  "test:ui": "vitest --ui",
  "test:coverage": "vitest --coverage",
  "test:ci": "vitest run --coverage"
}
```



Usage

Run Tests

```
# Run tests in watch mode
yarn test

# Run tests once
yarn test run

# Run tests with UI
yarn test:ui

# Generate coverage report
yarn test:coverage

# Run specific test file
yarn test button.test.tsx

# Run tests matching pattern
yarn test -t "should render"
```

Writing Tests

Example component test:

```
import { describe, it, expect, vi } from 'vitest';
import { render, screen, fireEvent } from '@testing-library/react';
import { MyComponent } from '@/components/MyComponent';

describe('MyComponent', () => {
  it('should render correctly', () => {
    render(<MyComponent text="Hello" />);
    expect(screen.getByText('Hello')).toBeInTheDocument();
  });

  it('should handle click', () => {
    const handleClick = vi.fn();
    render(<MyComponent onClick={handleClick} />);

    fireEvent.click(screen.getByRole('button'));
    expect(handleClick).toHaveBeenCalledTimes(1);
  });
});
```

Example service test:

```
import { describe, it, expect } from 'vitest';
import { myFunction } from '@/lib/myService';

describe('myService', () => {
  it('should process data correctly', () => {
    const result = myFunction('input');
    expect(result).toBe('expected output');
  });
});
```

Coverage Thresholds

The project is configured with the following coverage thresholds:

- Lines: 60%
- Functions: 60%
- Branches: 60%
- Statements: 60%

These thresholds will be enforced when running `yarn test:coverage`.

Best Practices

1. Test Organization

- Place tests next to the code they test OR in `__tests__` directory
- Use descriptive test names: `it('should do X when Y happens')`
- Group related tests with `describe` blocks

2. What to Test

- **Components:** Rendering, user interactions, conditional rendering
- **Services:** Business logic, data transformations, edge cases
- **Utils:** Pure functions, formatters, validators

3. What NOT to Test

- Third-party library internals
- Next.js framework internals
- Simple getters/setters
- Trivial code

4. Mocking

- Mock external dependencies (APIs, databases)
- Mock heavy computations
- Mock Date/Math.random for deterministic tests
- Use `vi.fn()` for function mocks
- Use `vi.mock()` for module mocks

5. Async Testing

```
it('should fetch data', async () => {
  const data = await fetchData();
  expect(data).toBeDefined();
});

// Or with waitFor
import { waitFor } from '@testing-library/react';

it('should show loading then data', async () => {
  render(<MyComponent />);

  expect(screen.getByText('Loading...')).toBeInTheDocument();

  await waitFor(() => {
    expect(screen.getByText('Data loaded')).toBeInTheDocument();
  });
});
```

Common Issues

Issue: “Cannot find module ‘@/...’”

Solution: Check that path aliases in `vitest.config.ts` match `tsconfig.json`

Issue: “window is not defined”

Solution: Ensure `environment: 'jsdom'` is set in `vitest.config.ts`

Issue: “useRouter() is not mocked”

Solution: The router is already mocked in `vitest.setup.ts`. If you need custom behavior, override it in your test file.

Issue: “Tests fail but code works”

Solution: Check that your test environment matches your runtime environment. You may need to add specific mocks.

Resources

- [Vitest Documentation](https://vitest.dev/) (<https://vitest.dev/>)
- [Testing Library Docs](https://testing-library.com/docs/react-testing-library/intro/) (<https://testing-library.com/docs/react-testing-library/intro/>)
- [Jest DOM Matchers](https://github.com/testing-library/jest-dom) (<https://github.com/testing-library/jest-dom>)

Next Steps

1. Manually add test scripts to `package.json`
2. Run `yarn test` to verify setup
3. Write tests for your critical components and services
4. Gradually increase test coverage
5. Add tests to CI/CD pipeline
6. Consider adding E2E tests with Playwright (already configured)



Examples Included

Four example test files have been created to demonstrate:

- Component testing (Button, KPICard)
- Service testing (sanitize functions)
- Utility testing (className merger)

You can use these as templates for writing your own tests.