

# Code Coverage Report

Unit Testing

Erik Conradie - 221147





# 01 Introduction

# Introduction

Welcome to the Code Coverage Report for Budget Buddy, a dynamic web application designed to help users manage their monthly finances by tracking salaries, expenses, and calculating tax brackets, savings, and disposable income. This report outlines my rigorous testing methodology, provides insights into my project structure, and showcases the effectiveness of my testing practices through detailed code coverage metrics.



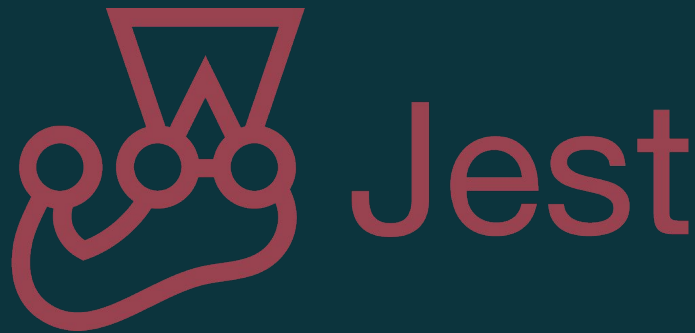


02

# Testing Strategy

# Testing Strategy

My testing strategy encompasses a multi-layered approach to ensure Budget Buddy's reliability and performance. I adopted Jest, a powerful testing framework, to facilitate unit, functionality, and component testing. This strategy ensures comprehensive coverage, from individual functions to complex user interactions, highlighting my commitment to delivering a robust and error-free application.





# 03 Project Structure

# Project Structure

Budget Buddy is structured to promote modularity and ease of testing. The project is divided into several key directories:

- `/src`: Contains all source code, including components and utilities.
- `/tests`: Houses our Jest test suites, organized by mainly functionality tests.
- `/Components`: Houses all Jest test suites for each component of the application.
- `/Snapshots`: Stores all the snapshots of each component.
- `/functions`: Within this file is each functionality of the application.
- `/items`: This is where each component's card and row component is stored.
- `/config`: Contains configuration files for testing environments and other setup requirements.



04

# Testing Environment



# Testing Environment

To create a conducive environment for testing, I configured Jest alongside supportive libraries like react-testing-library for component tests. My setup includes:

- **Jest Configuration:** Tailored to support JSX and ES6 modules, ensuring seamless testing of React components.
- **Mocking Strategy:** Utilizes Jest's mocking capabilities to simulate external dependencies and user interactions.
- **Babel Configuration:** Ensures the import of functions and components to my testing environment.
- **Vite:** Setup of the development environment for the React framework.



# 05 Unit Tests - Functionality

# Income Unit Test

2

Tests

## Adding new Income

Test for adding a new Income to our Income list.

- Correctly adding of a new Income to the array.
- New Income added has correct values.

 ▼

Erik

30000

Add

# Expenses Unit Test

2

Tests

## Adding new Expense

Test for adding a new Expense to our expenses list.

- Correctly adding of a new expense to the array.
- New expense added has correct values.

# Savings Unit Test

3

Tests

## Calculate Savings

Test for calculating the savings correctly.

- Correctly calculate savings amount from income.
- Handle the state of the savings with an empty income array.
- Handle the state of the savings when no percentage has been selected.

 Erik <b>R 3000.00</b> / R 30000.00	 Erik <b>R 10000.00</b> / R 100000.00	 Erik <b>R 30000.00</b> / R 300000.00
---	---	---

# Taxes Unit Test

7

Tests

## Calculate Tax Bracket

Test for calculating the tax bracket correctly.

- \*6\* Tests to return the correct tax bracket percentage depending on the income amount.

## Calculate Tax Amount

Test for calculating the tax amount correctly.

- Calculate the correct tax amount for the various tax brackets.



Erik

18%

- R 5400.00



Erik

36%

- R 36000.00



Erik

39%

- R 117000.00

# Totals Unit Test

8

Tests

## Total Income Before Tax

Test for calculating if the income is correct before the tax is applied.

- Calculate total income before tax correctly.
- Return correct value when Income is an empty array.

## Total Income After Tax

Test for calculating if the income is correct after the tax is applied.

- Calculate total income after tax correctly
- Return correct value when Income is an empty array.

# Totals Unit Test

## Total Expenses

Test for calculating if the total expenses amount is correct.

- Calculate total expenses correctly.
- Return correct value when expenses is an empty array.

## Left to spend Total

Test for calculating if the the amount left to spend is correct.

- Calculate left to spend correctly.
- Calculate correctly with no expenses and savings provided.

8

Tests



# Totals Unit Test

## Total Expenses

Test for calculating if the total expenses amount is correct.

## Left to spend Total

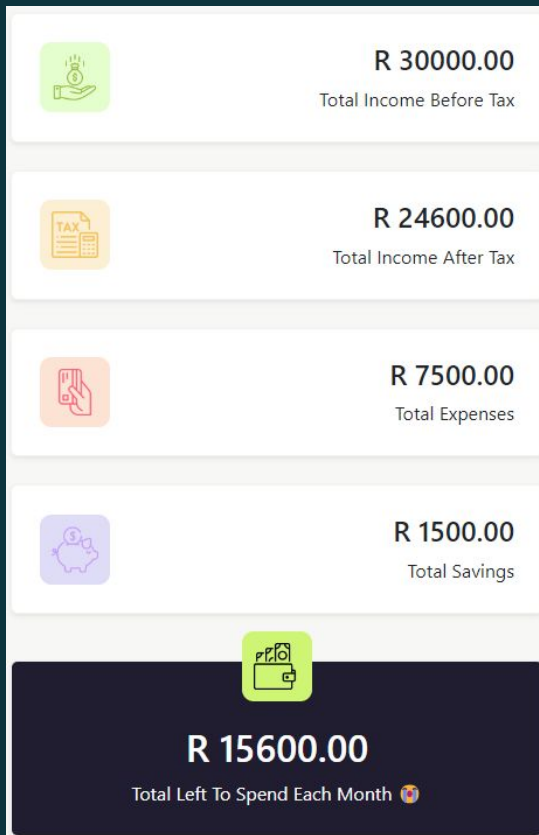
Test for calculating if the the amount left to spend is correct.

## Total Income Before Tax

Test for calculating if the income is correct before the tax is applied.

## Total Income After Tax

Test for calculating if the income is correct after the tax is applied.



8  
Tests



06

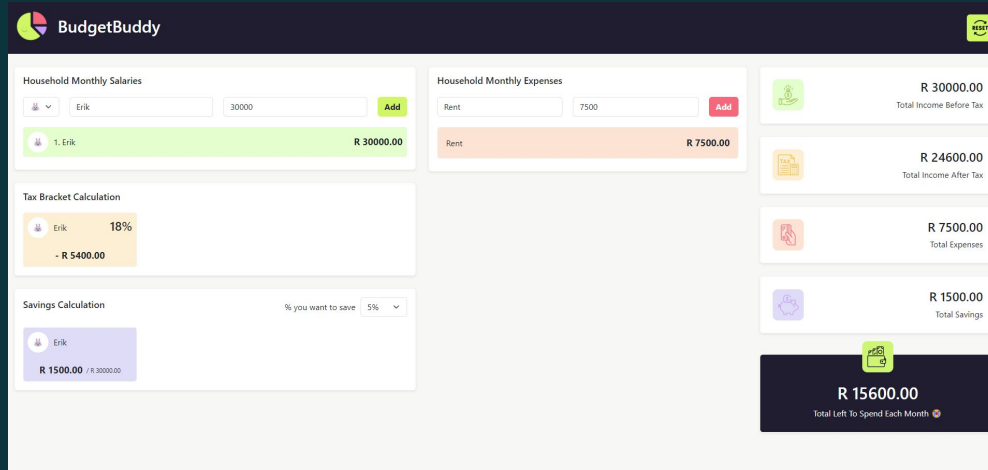
# Unit Tests - Components

# App Component Test

## Correctly Render App

Test for checking if the whole app renders correctly.

- Snapshot of app when rendered for the first time.



1  
Tests

# Expenses Component Test

## Correctly Render Expense

Test for checking if the expenses cards and component render correctly, as well as when a new expense is added.

- Test default state of expenses component.
- Test if expense card displays when 2 expenses are added.
- Test adding of new expense.

### Household Monthly Expenses

Add

Rent

R 7500.00

3

Tests



# Income Component Test

## Correctly Render Income

Test for checking if the income cards and component render correctly, as well as when a new income is added.

- Test default state of income component.
- Test if income card displays when 2 new incomes are added.
- Test adding of new income.


### Household Monthly Salaries

Erik

30000

Add



1. Erik

R 30000.00

3  
Tests

# Savings Component Test

3

Tests

## Correctly Render Savings

Test for checking if the savings cards and component render correctly, as well as when a new savings are added.

- Test default state of Savings component.
- Test if income card displays when 2 new savings are added.
- Test if savings component renders correctly.

Savings Calculation

% you want to save

5%



Erik

R 1500.00 / R 30000.00

# Taxes Component Test

2

Tests

## Correctly Render Taxes

Test for checking if the tax cards and component render correctly, as well as when a new tax is added.

- Test default state of Taxes component.
- Test if tax card displays when 2 new tax entries are added.

### Tax Bracket Calculation

 Erik 18%  
- R 5400.00



# Demo Walkthrough





# 08 Test Results

22

Unit Tests

6

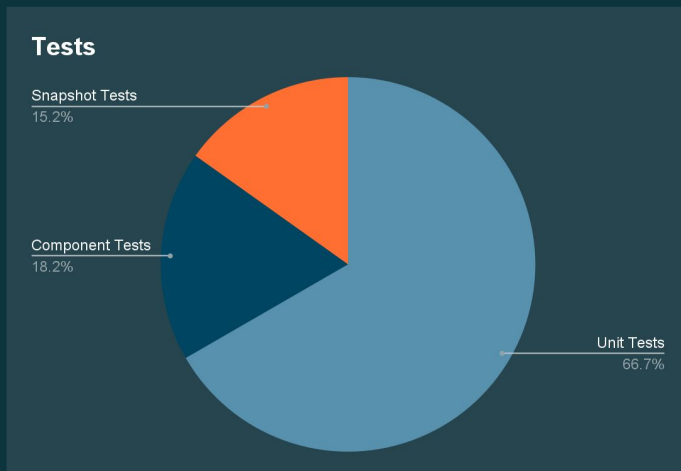
Component Tests

5

Snapshot Tests

100%

Pass Rate



```
Tests:      33 passed, 33 total
Snapshots:  5 passed, 5 total
Time:       5.263 s, estimated 15 s
```



# 09 Code Coverage Metrics



**94.26%**  
Statements

**86.95%**  
Functions

**88.88%**  
Branches

**94.01%**  
Lines



# Coverage Report

## All files

94.26% Statements 115/122 88.88% Branches 24/27 86.95% Functions 48/46 94.01% Lines 118/117

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File ▲		Statements ▾		Branches ▾		Functions ▾		Lines ▾	
src	<div><div></div></div>	82.6%	19/23	100%	0/0	57.14%	4/7	82.6%	19/23
src/components	<div><div></div></div>	94%	47/50	78.57%	11/14	84.21%	16/19	94%	47/50
src/components/functions	<div><div></div></div>	100%	37/37	100%	13/13	100%	14/14	100%	32/32
src/components/items	<div><div></div></div>	100%	12/12	100%	0/0	100%	6/6	100%	12/12



# 10 Challenges and Solutions

# Challenges and Solutions

Throughout the testing process, I encountered and overcame several challenges:

- Coverage Percentage: It took me a while to get the coverage outcomes to a decent percentage.
- UserEvent Testing: The user event testing was a challenge to get working.



# 11 Future Improvements



# Future Improvements

Plans for enhancing Budget Buddy's testing strategy include:

- **Integration Testing:** Expanding coverage to include end-to-end workflow tests.
- **Performance Testing:** Implementing tests to evaluate and optimize application speed and responsiveness.
- **Improved Coverage:** Plan to improve the coverage outcome to 100%.



# 12 Conclusion



# Conclusion

The comprehensive testing strategy and diligent implementation of code coverage practices for Budget Buddy have significantly contributed to its reliability and efficiency. Moving forward, I am committed to maintaining high testing standards, ensuring that Budget Buddy continues to serve as a trusted tool for personal finance management.



**Thank you!**

