Test Registration Authenticated

test_registration_authenticated: Verify that a new user can successfully register and is redirected to the login page.

Severity	Priority	Type	Layer
— Not set	— Not set	Integration	API
Milestone	**Automation**	**Status**	**Is Muted**
-	Should be automated	Actual	No

Pre-conditions

The user does not already exist in the test database.

Post-conditions

The endpoint should return a 302 status code indicating a redirect to the login page.

Steps to reproduce

Step	1
Action	Send a POST request to the /auth/register endpoint with the following data
Input data	username: test_user password: testpass
Expected result	Response status code: 302

Test Registration Duplicate

test_registration_duplicate: Verify that attempting to register a user that already exists returns an appropriate error.

Severity	**Priority**	**Type**	**Layer**
— Not set	— Not set	Integration	API
Milestone	**Automation**	**Status**	**Is Muted**
-	Should be automated	Actual	No

Pre-conditions

The user already exists in the test database.

Post-conditions

The endpoint should return a 400 status code with a message indicating that the user already exists.

Steps to reproduce

Step	1
Action	Insert a test user into the test database
Input data	username: test_user password: testpass
Expected result	

Step	2
Action	Send a POST request to the /auth/register endpoint with the following data
Input data	username: test_user password: testpass
Expected result	Response Status Code: 400 Response Message: User already exists

STP-14 / API Endpoints / Registration and Login Endpoints

Test Login Authenticated

test_login_authenticated: Verify that a registered user can successfully log in with valid credentials.

Severity	Priority	Type	Layer
— Not set	— Not set	Integration	API
Milestone	**Automation**	**Status**	**Is Muted**
-	Should be automated	Actual	No

Pre-conditions

The user exists in the test database with the correct credentials.

Post-conditions

The endpoint should return a 302 status code indicating a redirect to the dashboard page.

Steps to reproduce

Step	1
Action	insert a test user into the database
Input data	username: test_user password: testpass
Expected result	

Step	2
Action	send a POST request to /auth/login endpoint
Input data	username: test_user password: testpass
Expected result	Response Status Code: 302

Test Login Unauthorized

test_login_unauthorized: Verify that a login attempt with incorrect credentials returns an appropriate error.

Severity	**Priority**	**Type**	**Layer**
— Not set	— Not set	Integration	API
Milestone	**Automation**	**Status**	**Is Muted**
-	Should be automated	Actual	No

Pre-conditions

The user exists in the test database but the provided credentials are incorrect.

Post-conditions

The endpoint should return a 401 status code with a message indicating invalid credentials.

Steps to reproduce

Step	1
Action	Insert the test user into the database
Input data	username: test_user password: testpass
Expected result	

Step	2
Action	Send a POST request to the /auth/login endpoint with invalid credentials
Input data	username: test_user password: wrongpass
Expected result	Response Status Code: 401 Response Message: Invalid credentials.

STP-10

Verify Login using Selenium

Verify successful Budget Application Login using Selenium Tool

Severity	**Priority**	**Type**	**Layer**
○ Normal	— Not set	Other	Not set
Milestone	**Automation**	**Status**	**Is Muted**
-	Manual	Actual	No

Pre-conditions

Selenium and Python installed, and Budget Application is running

Steps to reproduce

Step	1
Action	Enter Login and Password
Input data	
Expected result	Verify URL after Login

Step	2
Action	Login
Input data	
Expected result	Verify Page title after Login

STP-11

Create Budget using Selenium

Create Budget using Selenium Web Tool

Severity	Priority	Type	Layer
○ Normal	— Not set	Other	Not set
Milestone	**Automation**	**Status**	**Is Muted**
-	Manual	Actual	No

Steps to reproduce

Step	1
Action	Login
Input data	
Expected result	Create Budget screen appears

Step	2
Action	Create budget and submit
Input data	
Expected result	Verify budget added to list

STP-5 / Model: Budget / create_budget

Add Budget

Check creating budget with valid input

Severity	**Priority**	**Type**	**Layer**
○ Normal	— Not set	Other	Not set
Milestone	**Automation**	**Status**	**Is Muted**
-	Manual	Actual	No

Post-conditions

budget was added to the database

STP-6 / Model: Budget / create_budget

Add invalid input

create a budget with invalid input

Severity	Priority	Type	Layer
○ Normal	— Not set	Other	Not set

Milestone	Automation	Status	Is Muted
-	Manual	Actual	No

Post-conditions

budget was not added to the database

STP-7 / Model: Budget / create_budget

Add invalid data

Verify no budget was added with invalid data

Severity	**Priority**	**Type**	**Layer**
○ Normal	— Not set	Other	Not set
Milestone	**Automation**	**Status**	**Is Muted**
-	Manual	Actual	No

Post-conditions

budget was not added to the database

STP-8 / Model: Budget / create_budget

Add budget with missing amount

Test creating a budget with missing amount field

Severity	**Priority**	**Type**	**Layer**
○ Normal	— Not set	Other	Not set
Milestone	**Automation**	**Status**	**Is Muted**
-	Manual	Actual	No

Post-conditions

budget was not added to the database

Test Generate Report Normal

test_generate_report_normal: Verify that a report is generated correctly for a budget with both incomes and expenses.

Severity	**Priority**	**Type**	**Layer**
— Not set	— Not set	Functional	Unit
Milestone	**Automation**	**Status**	**Is Muted**
-	Should be automated	Actual	No

Pre-conditions

The budget has both income and expense entries.

Post-conditions

The report should show the total income as $1700, total expenses as $1000, and savings as $700.

Steps to reproduce

Step	1
Action	Ensure budget has the following incomes
Input data	• Job: $1000 • Freelance: $500 • Side Hustle: $200
Expected result	

Step	2
Action	Ensure the budget has the following expenses:
Input data	• Utilities: $200 • Rent: $700 • Entertainment: $100
Expected result	

Step	3
Action	Generate the report
Input data	Budget: test_budget
Expected result	Total income: $1700 Total expenses: $1000 Savings: $700

Test Generate Report No Income

test_generate_report_no_income: Verify that a report is generated correctly for a budget with no incomes.

Severity	**Priority**	**Type**	**Layer**
— Not set	— Not set	Functional	Unit
Milestone	**Automation**	**Status**	**Is Muted**
-	Should be automated	Actual	No

Pre-conditions

The budget has expense entries but no income entries.

Post-conditions

The report should show the total income as $0, total expenses as $1000, and savings as -$1000.

Steps to reproduce

Step	1
Action	Ensure the budget has no incomes.
Input data	
Expected result	

Step	2
Action	Ensure the budget has the following expenses:
Input data	• Utilities: $200 • Rent: $700 • Entertainment: $100
Expected result	

Step	3
Action	Generate the report
Input data	Budget: test_budget
Expected result	Total Income: $0 Total Expenses: $1000 Savings: -$1000

Test Generate Report No Expense

test_generate_report_no_expense: Verify that a report is generated correctly for a budget with no expenses.

Severity	**Priority**	**Type**	**Layer**
— Not set	— Not set	Functional	Unit
Milestone	**Automation**	**Status**	**Is Muted**
-	Should be automated	Actual	No

Pre-conditions

The budget has income entries but no expense entries.

Post-conditions

The report should show the total income as $1700, total expenses as $0, and savings as $1700.

Steps to reproduce

Step	1
Action	Ensure the budget has the following incomes:
Input data	• Job: $1000 • Freelance: $500 • Side Hustle: $200
Expected result	

Step	2
Action	Ensure the budget has no expenses.
Input data	
Expected result	

Step	3
Action	Generate the report.
Input data	Budget: test_budget
Expected result	Total Income: $1700 Toal Expenses: $0 Savings: $1700

Test Generate Report Empty Input

test_generate_report_empty_input: Verify that a report is generated correctly for a budget with no incomes and no expenses.

Severity	**Priority**	**Type**	**Layer**
— Not set	— Not set	Functional	Unit
Milestone	**Automation**	**Status**	**Is Muted**
-	Should be automated	Actual	No

Pre-conditions

The budget has no income entries and no expense entries.

Post-conditions

The report should show the total income as $0, total expenses as $0, and savings as $0.

Steps to reproduce

Step	1
Action	Ensure the budget has no incomes.
Input data	
Expected result	

Step	2
Action	Ensure the budget has no expenses.
Input data	
Expected result	

Step	3
Action	Generate the report.
Input data	Budget: test_budget
Expected result	Total Income: $0 Total Expenses: $0 Savings: $0

Test Generate Report Mixed Input

test_generate_report_mixed_input: Verify that a report is generated correctly for a budget with mixed (integer and float) incomes and expenses.

Severity	Priority	Type	Layer
— Not set	— Not set	Functional	Unit
Milestone	**Automation**	**Status**	**Is Muted**
-	Should be automated	Actual	No

Pre-conditions

The budget has mixed (decimal and whole number) income and expense entries.

Post-conditions

The report should show the total income as $1700.50, total expenses as $700.75, and savings as $999.75.

Steps to reproduce

Step	1
Action	Ensure the budget has the following incomes:
Input data	• Job: $1500.50 • Side Hustle: $200
Expected result	

Step	2
Action	Ensure the budget has the following expenses:
Input data	• Utilities: $200.75 • Rent: $500
Expected result	

Step	3
Action	Generate the report.
Input data	Budget: test_budget
Expected result	Total Income: $1700.50 Total Expenses: $700.75 Savings: $999.75

Test Calculate Savings Normal

test_calculate_savings_normal: Verify that savings are calculated correctly for a budget with both incomes and expenses.

Severity	Priority	Type	Layer
— Not set	— Not set	Functional	Unit
Milestone	**Automation**	**Status**	**Is Muted**
-	Should be automated	Actual	No

Pre-conditions

The budget has both income and expense entries.

Post-conditions

The savings should be $800 (income - expenses).

Steps to reproduce

Step	1
Action	Ensure the budget has the following incomes:
Input data	Job: $1500
Expected result	

Step	2
Action	Ensure the budget has the following expenses:
Input data	• Utilities: $200 • Rent: $500
Expected result	

Step	3
Action	Calculate the savings.
Input data	
Expected result	$800

STP-22 / Model: Budget / Test Calculate Savings

Test Calculate Savings No Income

test_calculate_savings_no_income: Verify that savings are calculated correctly for a budget with no incomes.

Severity	**Priority**	**Type**	**Layer**
— Not set	— Not set	Functional	Unit
Milestone	**Automation**	**Status**	**Is Muted**
-	Should be automated	Actual	No

Pre-conditions

The budget has expense entries but no income entries.

Post-conditions

The savings should be -$700 (income - expenses).

Steps to reproduce

Step	1
Action	Ensure the budget has no incomes.
Input data	
Expected result	

Step	2
Action	Ensure the budget has the following expenses:
Input data	• Utilities: $200 • Rent: $500
Expected result	

Step	3
Action	Calculate the savings.
Input data	
Expected result	-$700

Test Calculate Savings No Expenses

test_calculate_savings_no_expenses: Verify that savings are calculated correctly for a budget with no expenses.

Severity	Priority	Type	Layer
— Not set	— Not set	Functional	Unit
Milestone	**Automation**	**Status**	**Is Muted**
-	Should be automated	Actual	No

Pre-conditions

The budget has income entries but no expense entries.

Post-conditions

The savings should be $1500 (income - expenses).

Steps to reproduce

Step	1
Action	Ensure the budget has the following incomes:
Input data	Job: $1500
Expected result	

Step	2
Action	Ensure the budget has no expenses.
Input data	
Expected result	

Step	3
Action	Calculate the savings.
Input data	
Expected result	$1500

Test Calculate Savings Empty Input

test_calculate_savings_empty_input: Verify that savings are calculated correctly for a budget with no incomes and no expenses.

Severity	**Priority**	**Type**	**Layer**
— Not set	— Not set	Functional	Unit
Milestone	**Automation**	**Status**	**Is Muted**
-	Should be automated	Actual	No

Pre-conditions

The budget has no income entries and no expense entries.

Post-conditions

The savings should be $0 (income - expenses).

Steps to reproduce

Step	1
Action	Ensure the budget has no incomes.
Input data	
Expected result	

Step	2
Action	Ensure the budget has no expenses.
Input data	
Expected result	

Step	3
Action	Calculate the savings.
Input data	$0
Expected result	

STP-25 / Model: Budget / Test Calculate Savings

Test Calculate Savings Mixed Input

test_calculate_savings_mixed_input: Verify that savings are calculated correctly for a budget with mixed incomes and expenses.

Severity	**Priority**	**Type**	**Layer**
— Not set	— Not set	Functional	Unit
Milestone	**Automation**	**Status**	**Is Muted**
-	Should be automated	Actual	No

Pre-conditions

The budget has mixed (decimal and whole number) income and expense entries.

Post-conditions

The savings should be $999.75 (income - expenses).

Steps to reproduce

Step	1
Action	Ensure the budget has the following incomes:
Input data	• Job: $1500.50 • Side Hustle: $200
Expected result	

Step	2
Action	Ensure the budget has the following expenses:
Input data	• Utilities: $200.75 • Rent: $500
Expected result	

Step	3
Action	Calculate the savings.
Input data	
Expected result	$999.75

STP-26 / Model: Budget / Test Calculate Savings

Test Calculate Savings Negative Input

test_calculate_savings_negative_input: Verify that savings are calculated correctly for a budget with negative income and expense entries.

Severity	**Priority**	**Type**	**Layer**
○ Normal	— Not set	Functional	Not set
Milestone	**Automation**	**Status**	**Is Muted**
-	Should be automated	Actual	No

Pre-conditions

The budget has negative income and/or expense entries.

Post-conditions

The savings should be -$1200 (income - expenses).

Steps to reproduce

Step	1
Action	Ensure the budget has the following incomes:
Input data	• Job: -$1500
Expected result	

Step	2
Action	Ensure the budget has the following expenses:
Input data	• Utilities: $200 • Rent: -$500
Expected result	

Step	3
Action	Calculate the savings.
Input data	
Expected result	-$1200

Test Calculate Savings Multiple Steps

test_calculate_savings_multiple_steps: Verify that savings are calculated correctly when incomes and expenses are added incrementally.

Severity	Priority	Type	Layer
— Not set	— Not set	Functional	Unit
Milestone	**Automation**	**Status**	**Is Muted**
-	Should be automated	Actual	No

Pre-conditions

The budget starts with no income and no expense entries.

Post-conditions

The savings should be $1175 (income - expenses).

Steps to reproduce

Step	1
Action	Ensure the budget starts with no incomes and no expenses.
Input data	
Expected result	

Step	2
Action	Add the following incomes and expenses incrementally:
Input data	• Add income: Job: $1500 • Add income: Side Hustle: $200 • Add expense: Utilities: $275 • Add income: Freelance: $700 • Add expense: Rent: $950
Expected result	

Step	3
Action	Calculate the savings.
Input data	
Expected result	$1175

STP-44 / Model: Budget / Test Remove Income and Expenses

Test Remove Single Income

test_remove_single_income: Verify that a single income entry can be removed from the budget.

Severity	**Priority**	**Type**	**Layer**
— Not set	— Not set	Functional	Unit
Milestone	**Automation**	**Status**	**Is Muted**
-	Should be automated	Actual	No

Pre-conditions

The budget has multiple income entries.

Post-conditions

The incomes list should contain {'source': 'Freelance', 'amount': 500} and {'source': 'Side Hustle', 'amount': 200}.

Steps to reproduce

Step	1
Action	Ensure the budget has the following incomes:
Input data	• Job: $1000 • Freelance: $500 • Side Hustle: $200
Expected result	

Step	2
Action	Remove the income with
Input data	source: 'Job' amount: $1000
Expected result	

Step	3
Action	Check the incomes list in the budget.
Input data	
Expected result	{'source': 'Freelance', 'amount': 500} and {'source': 'Side Hustle', 'amount': 200}

Test Remove Multiple Incomes

test_remove_multiple_incomes: Verify that multiple income entries can be removed from the budget.

Severity	**Priority**	**Type**	**Layer**
— Not set	— Not set	Functional	Unit
Milestone	**Automation**	**Status**	**Is Muted**
-	Should be automated	Actual	No

Pre-conditions

The budget has multiple income entries.

Post-conditions

The incomes list should contain {'source': 'Side Hustle', 'amount': 200}.

Steps to reproduce

Step	1
Action	Ensure the budget has the following incomes:
Input data	• Job: $1000 • Freelance: $500 • Side Hustle: $200
Expected result	

Step	2
Action	Remove the income with
Input data	source: 'Job' amount: $1000
Expected result	

Step	3
Action	Remove the income with
Input data	source: 'Freelance' amount: $500
Expected result	

Step	4
Action	Check the incomes list in the budget.
Input data	
Expected result	{'source': 'Side Hustle', 'amount': 200}

Test Remove Single Expense

test_remove_single_expense: Verify that a single expense entry can be removed from the budget.

Severity	Priority	Type	Layer
— Not set	— Not set	Functional	Unit
Milestone	**Automation**	**Status**	**Is Muted**
-	Should be automated	Actual	No

Pre-conditions

The budget has multiple expense entries.

Post-conditions

The expenses list should contain {'category': 'Utilities', 'amount': 200} and {'category': 'Entertainment', 'amount': 100}

Steps to reproduce

Step	1
Action	Ensure the budget has the following expenses:
Input data	• Utilities: $200 • Rent: $700 • Entertainment: $100
Expected result	

Step	2
Action	Remove the expense with
Input data	category: 'Rent' amount: $700
Expected result	

Step	3
Action	Check the expenses list in the budget.
Input data	
Expected result	{'category': 'Utilities', 'amount': 200} and {'category': 'Entertainment', 'amount': 100}

Test Remove Multiple Expenses

test_remove_multiple_expenses: Verify that multiple expense entries can be removed from the budget.

Severity	**Priority**	**Type**	**Layer**
— Not set	— Not set	Functional	Unit
Milestone	**Automation**	**Status**	**Is Muted**
-	Should be automated	Actual	No

Pre-conditions

The budget has multiple expense entries.

Post-conditions

The expenses list should contain {'category': 'Entertainment', 'amount': 100}

Steps to reproduce

Step	1
Action	Ensure the budget has the following expenses:
Input data	• Utilities: $200 • Rent: $700 • Entertainment: $100
Expected result	

Step	2
Action	Remove the expense with
Input data	category: 'Rent' amount: $700
Expected result	

Step	3
Action	Remove the expense with
Input data	category: 'Utilities' amount: $200
Expected result	

Step	4
Action	Check the expenses list in the budget.
Input data	
Expected result	{'category': 'Entertainment', 'amount': 100}

STP-48 / Model: Budget / Test Remove Income and Expenses

Test Remove Incomes and Expenses

test_remove_incomes_and_expenses: Verify that both income and expense entries can be removed from the budget.

Severity	**Priority**	**Type**	**Layer**
— Not set	— Not set	Functional	Unit
Milestone	**Automation**	**Status**	**Is Muted**
-	Should be automated	Actual	No

Pre-conditions

The budget has multiple income and expense entries.

Post-conditions

The incomes list should contain {'source': 'Freelance', 'amount': 500} and {'source': 'Side Hustle', 'amount': 200}. The expenses list should contain {'category': 'Utilities', 'amount': 200} and {'category': 'Entertainment', 'amount': 100}.

Steps to reproduce

Step	1
Action	Ensure the budget has the following incomes:
Input data	• Job: $1000 • Freelance: $500 • Side Hustle: $200
Expected result	

Step	2
Action	Ensure the budget has the following expenses:
Input data	• Utilities: $200 • Rent: $700 • Entertainment: $100
Expected result	

Step	3
Action	Remove the income with
Input data	source: 'Job' amount: $1000
Expected result	

Step	4
Action	Remove the expense with
Input data	category: 'Rent' amount: $700

Expected result	

Step	5
Action	Check the incomes list in the budget.
Input data	
Expected result	{'source': 'Freelance', 'amount': 500} and {'source': 'Side Hustle', 'amount': 200}

Step	6
Action	Check the expenses list in the budget.
Input data	
Expected result	{'category': 'Utilities', 'amount': 200} and {'category': 'Entertainment', 'amount': 100}

STP-28 / Model: Budget / Test Add Income

Test Add Income

test_add_income: Verify that an income entry can be added to the budget.

Severity	**Priority**	**Type**	**Layer**
— Not set	— Not set	Functional	Unit
Milestone	**Automation**	**Status**	**Is Muted**
-	Should be automated	Actual	No

Pre-conditions

The budget exists.

Post-conditions

The incomes list should contain one entry with {'source': 'Job', 'amount': 1000}

Steps to reproduce

Step	1
Action	Ensure the budget is initialized.
Input data	
Expected result	

Step	2
Action	Add income
Input data	source: 'Job' amount: $1000
Expected result	

Step	3
Action	Check the incomes list in the budget.
Input data	
Expected result	{'source': 'Job', 'amount': 1000}

Test Add Income Zero Amount

test_add_income_zero_amount: Verify that an income entry with a zero amount is not added to the budget.

Severity	**Priority**	**Type**	**Layer**
— Not set	— Not set	Functional	Unit
Milestone	**Automation**	**Status**	**Is Muted**
-	Should be automated	Actual	No

Pre-conditions

The budget exists.

Post-conditions

The incomes list should be empty.

Steps to reproduce

Step	1
Action	Ensure the budget is initialized.
Input data	
Expected result	

Step	2
Action	Add income
Input data	source: 'Job' amount: $0
Expected result	

Step	3
Action	Check the incomes list in the budget.
Input data	incomes = []
Expected result	

Test Add Income Negative Amount

test_add_income_negative_amount: Verify that an income entry with a negative amount is not added to the budget.

Severity	**Priority**	**Type**	**Layer**
— Not set	— Not set	Functional	Unit
Milestone	**Automation**	**Status**	**Is Muted**
-	Should be automated	Actual	No

Pre-conditions

The budget exists.

Post-conditions

The incomes list should be empty.

Steps to reproduce

Step	1
Action	Ensure the budget is initialized.
Input data	
Expected result	

Step	2
Action	Add income
Input data	source: 'Gift' amount: -$500
Expected result	

Step	3
Action	Check the incomes list in the budget.
Input data	
Expected result	incomes = []

Test Add Multiple Incomes

test_add_multiple_incomes: Verify that multiple income entries can be added to the budget.

Severity	**Priority**	**Type**	**Layer**
— Not set	— Not set	Functional	Unit
Milestone	**Automation**	**Status**	**Is Muted**
-	Should be automated	Actual	No

Pre-conditions

The budget exists.

Post-conditions

The incomes list should contain two entries: {'source': 'Job', 'amount': 1000} and {'source': 'Freelance', 'amount': 500}

Steps to reproduce

Step	1
Action	Ensure the budget is initialized.
Input data	
Expected result	

Step	2
Action	Add income
Input data	source: 'Job' amount: $1000
Expected result	

Step	3
Action	Add income
Input data	source: 'Freelance' amount: $500
Expected result	

Step	4
Action	Check the incomes list in the budget.
Input data	
Expected result	{'source': 'Job', 'amount': 1000} and {'source': 'Freelance', 'amount': 500}

Test Add Income Large Amount

test_add_income_large_amount: Verify that a large income entry can be added to the budget.

Severity	**Priority**	**Type**	**Layer**
— Not set	— Not set	Functional	Unit
Milestone	**Automation**	**Status**	**Is Muted**
-	Should be automated	Actual	No

Pre-conditions

The budget exists.

Post-conditions

The incomes list should contain one entry with {'source': 'Investment', 'amount': 1,000,000,000}.

Steps to reproduce

Step	1
Action	Ensure the budget is initialized.
Input data	
Expected result	

Step	2
Action	Add income
Input data	source: 'Investment' amount: $1,000,000,000
Expected result	

Step	3
Action	Check the incomes list in the budget
Input data	
Expected result	{'source': 'Investment', 'amount': 1,000,000,000}

Test Add Income Float Amount

test_add_income_float_amount: Verify that an income entry with a float amount can be added to the budget.

Severity	**Priority**	**Type**	**Layer**
— Not set	— Not set	Functional	Unit
Milestone	**Automation**	**Status**	**Is Muted**
-	Should be automated	Actual	No

Pre-conditions

The budget exists.

Post-conditions

The incomes list should contain one entry with {'source': 'Tips', 'amount': 0.01}

Steps to reproduce

Step	1
Action	Ensure the budget is initialized.
Input data	
Expected result	

Step	2
Action	Add income
Input data	source: 'Tips' amount: $0.01
Expected result	

Step	3
Action	Check the incomes list in the budget.
Input data	{'source': 'Tips', 'amount': 0.01}
Expected result	

Test Add Income Empty Amount

test_add_income_empty_amount: Verify that adding an income entry with an empty amount raises aValueError

Severity	**Priority**	**Type**	**Layer**
— Not set	— Not set	Functional	Unit
Milestone	**Automation**	**Status**	**Is Muted**
-	Should be automated	Actual	No

Pre-conditions

The budget exists.

Post-conditions

A ValueError should be raised.

Steps to reproduce

Step	1
Action	Ensure the budget is initialized.
Input data	
Expected result	

Step	2
Action	Attempt to add income with
Input data	source: 'Job' amount: ''
Expected result	

Step	3
Action	Check that a ValueError is raised.
Input data	
Expected result	ValueError("Amount cannot be empty.")

Test Add Income Empty Source

test_add_income_empty_source: Verify that adding an income entry with an empty source raises aValueError.

Severity	**Priority**	**Type**	**Layer**
— Not set	— Not set	Functional	Unit
Milestone	**Automation**	**Status**	**Is Muted**
-	Should be automated	Actual	No

Pre-conditions

The budget exists.

Post-conditions

A ValueError should be raised.

Steps to reproduce

Step	1
Action	Ensure the budget is initialized.
Input data	
Expected result	

Step	2
Action	Attempt to add income with
Input data	source: '' amount: $200
Expected result	

Step	3
Action	Check that a ValueError is raised.
Input data	
Expected result	ValueError("Income source must be provided.")

Test Add Income None Amount

test_add_income_none_amount: Verify that adding an income entry with aNone amount raises a ValueError.

Severity	**Priority**	**Type**	**Layer**
— Not set	— Not set	Functional	Unit
Milestone	**Automation**	**Status**	**Is Muted**
-	Should be automated	Actual	No

Pre-conditions

The budget exists.

Post-conditions

A ValueError should be raised.

Steps to reproduce

Step	1
Action	Ensure the budget is initialized.
Input data	
Expected result	

Step	2
Action	Attempt to add income with
Input data	source: 'Job' amount: None
Expected result	

Step	3
Action	Check that a ValueError is raised.
Input data	
Expected result	ValueError("Amount cannot be empty.")

Test Add Income String Amount

test_add_income_string_amount: Verify that adding an income entry with a string amount raises aTypeError.

Severity	Priority	Type	Layer
— Not set	— Not set	Functional	Unit
Milestone	**Automation**	**Status**	**Is Muted**
-	Should be automated	Actual	No

Pre-conditions

The budget exists.

Post-conditions

A TypeError should be raised.

Steps to reproduce

Step	1
Action	Ensure the budget is initialized.
Input data	
Expected result	

Step	2
Action	Attempt to add income with the source
Input data	source: 'Job' amount: 'Pancakes'
Expected result	

Step	3
Action	Check that a `TypeError` is raised.
Input data	
Expected result	TypeError("Amount must be a number.")

Test Add Expense Valid

testAddExpense: Tests whether adding a valid expense is correctly reflected in the budget's expense list.

Severity	Priority	Type	Layer
— Not set	— Not set	Functional	Unit
Milestone	**Automation**	**Status**	**Is Muted**
-	Should be automated	Actual	No

Pre-conditions

- A Budget object is instantiated with the name "test_budget".
- The expense list is initially empty.

Post-conditions

The expense list contains the newly added expense with the specified category and amount.

Steps to reproduce

Step	1
Action	Create a Budget
Input data	"test_budget"
Expected result	

Step	2
Action	Add an expense
Input data	category, "Utilities" amount, 150
Expected result	

Step	3
Action	Verify that the expense list contains the added expense
Input data	
Expected result	{'category': 'Utilities', 'amount': 150}

Test Add Expense Zero Amount

testAddExpenseZero: Tests whether adding an expense with a zero amount is ignored (i.e., not added to the expense list).

Severity	**Priority**	**Type**	**Layer**
— Not set	— Not set	Functional	Unit
Milestone	**Automation**	**Status**	**Is Muted**
-	Should be automated	Actual	No

Pre-conditions

- A Budget object is instantiated with the name "test_budget".
- The expense list is initially empty.

Post-conditions

The expense list remains empty after attempting to add a zero amount expense.

Steps to reproduce

Step	1
Action	Create a Budget instance
Input data	"test_budget"
Expected result	

Step	2
Action	Add an expense with a zero amount
Input data	category, 'Food' amount, 0
Expected result	

Step	3
Action	Verify that the expense list remains empty
Input data	
Expected result	[ ]

STP-51 / Model: Budget / Test Add Expense

Test Add Expense Negative Amount

testAddExpenseNegative: Tests whether adding an expense with a negative amount is ignored (i.e., not added to the expense list).

Severity	**Priority**	**Type**	**Layer**
— Not set	— Not set	Functional	Unit
Milestone	**Automation**	**Status**	**Is Muted**
-	Should be automated	Actual	No

Pre-conditions

- A Budget object is instantiated with the name "test_budget".
- The expense list is initially empty.

Post-conditions

The expense list remains empty after attempting to add a negative amount expense.

Steps to reproduce

Step	1
Action	Create a Budget instance
Input data	'test_budget'
Expected result	

Step	2
Action	Add an expense with a negative amount
Input data	category, 'Gas' amount, -50
Expected result	

Step	3
Action	Verify that the expense list remains empty
Input data	
Expected result	[ ]

Test Calculate Total Incomes Single

test_calculate_total_incomes_single: Verify that the total income is calculated correctly when there is a single income entry.

Severity	**Priority**	**Type**	**Layer**
— Not set	— Not set	Functional	Unit
Milestone	**Automation**	**Status**	**Is Muted**
-	Should be automated	Actual	No

Pre-conditions

The budget has a single income entry.

Post-conditions

The total income should be $1500.

Steps to reproduce

Step	1
Action	Ensure the budget has the following income:
Input data	Job: $1500
Expected result	

Step	2
Action	Calculate the total income.
Input data	
Expected result	

Step	3
Action	Check the result.
Input data	
Expected result	$1500

Test Calculate Total Incomes Multiple

test_calculate_total_incomes_multiple: Verify that the total income is calculated correctly when there are multiple income entries.

Severity	**Priority**	**Type**	**Layer**
— Not set	— Not set	Functional	Unit
Milestone	**Automation**	**Status**	**Is Muted**
-	Should be automated	Actual	No

Pre-conditions

The budget has multiple income entries.

Post-conditions

The total income should be $1700.

Steps to reproduce

Step	1
Action	Ensure the budget has the following incomes:
Input data	• Job: $1500 • Side Hustle: $200
Expected result	

Step	2
Action	Calculate the total income.
Input data	
Expected result	

Step	3
Action	Check the result.
Input data	
Expected result	$1700

Test Calculate Total Incomes Empty

test_calculate_total_incomes_empty: Verify that the total income is calculated correctly when there are no income entries.

Severity	**Priority**	**Type**	**Layer**
— Not set	— Not set	Functional	Unit
Milestone	**Automation**	**Status**	**Is Muted**
-	Should be automated	Actual	No

Pre-conditions

The budget has no income entries.

Post-conditions

The total income should be $0.

Steps to reproduce

Step	1
Action	Ensure the budget has no incomes.
Input data	
Expected result	

Step	2
Action	Calculate the total income.
Input data	
Expected result	

Step	3
Action	Check the result.
Input data	
Expected result	$0

Test Calculate Total Expenses Single

test_calculate_total_expenses_single: Verify that the total expenses are calculated correctly when there is a single expense entry.

Severity	**Priority**	**Type**	**Layer**
— Not set	— Not set	Functional	Unit
Milestone	**Automation**	**Status**	**Is Muted**
-	Should be automated	Actual	No

Pre-conditions

The budget has a single expense entry.

Post-conditions

The total expenses should be $200.

Steps to reproduce

Step	1
Action	Ensure the budget has the following expense:
Input data	Utilities: $200
Expected result	

Step	2
Action	Calculate the total expenses.
Input data	
Expected result	

Step	3
Action	Check the result.
Input data	
Expected result	$200

Test Calculate Total Expenses Multiple

test_calculate_total_expenses_multiple: Verify that the total expenses are calculated correctly when there are multiple expense entries.

Severity	**Priority**	**Type**	**Layer**
— Not set	— Not set	Functional	Unit
Milestone	**Automation**	**Status**	**Is Muted**
-	Should be automated	Actual	No

Pre-conditions

The budget has multiple expense entries.

Post-conditions

The total expenses should be $700.

Steps to reproduce

Step	1
Action	Ensure the budget has the following expenses:
Input data	• Utilities: $200 • Rent: $500
Expected result	

Step	2
Action	Calculate the total expenses.
Input data	
Expected result	

Step	3
Action	Check the result.
Input data	
Expected result	$700

Test Calculate Total Expenses Empty

test_calculate_total_expenses_empty: Verify that the total expenses are calculated correctly when there are no expense entries.

Severity	**Priority**	**Type**	**Layer**
— Not set	— Not set	Functional	Unit
Milestone	**Automation**	**Status**	**Is Muted**
-	Should be automated	Actual	No

Pre-conditions

The budget has no expense entries.

Post-conditions

The total expenses should be $0.

Steps to reproduce

Step	1
Action	Ensure the budget has no expenses.
Input data	
Expected result	

Step	2
Action	Calculate the total expenses.
Input data	
Expected result	

Step	3
Action	Check the result.
Input data	
Expected result	$0

STP-52

Add Expense to Budget

This test case verifies that a user can successfully add a new expense to their budget and that the expense is correctly reflected in the expense list.

Severity	Priority	Type	Layer
— Not set	— Not set	Acceptance	Unit
Milestone	**Automation**	**Status**	**Is Muted**
-	Manual	Actual	No

Pre-conditions

- The user has an existing budget named "test_budget".
- The expense list for "test_budget" is initially empty.

Post-conditions

- The expense list for "test_budget" contains the newly added expense.
- The total budget reflects the addition of the new expense.

Steps to reproduce

Step	1
Action	The user logs into the budget application
Input data	'test_user' 'testpass'
Expected result	

Step	2
Action	The user navigates to 'test_budget'
Input data	
Expected result	

Step	3
Action	User selects expense category
Input data	'Groceries'
Expected result	

Step	4
Action	User enters expense amount
Input data	100
Expected result	

Step	5
Action	User clicks "Add Expense"
Input data	
Expected result	

Step	6
Action	User views list of expenses
Input data	
Expected result	Expenses by Category: Groceries: $150.0

Step	7
Action	User views total expense amount
Input data	
Expected result	Total Expenses: $150.0

STP-53

Add Income to Budget

This test case verifies that a user can successfully add a new income to their budget and that the income is correctly reflected in the income list.

Severity	**Priority**	**Type**	**Layer**
— Not set	— Not set	Acceptance	Unit
Milestone	**Automation**	**Status**	**Is Muted**
-	Manual	Actual	No

Pre-conditions

- The user has an existing budget named "test_budget".
- The income list for "test_budget" is initially empty.

Post-conditions

- The income list for "test_budget" contains the newly added income.
- The total budget reflects the addition of the new income.

Steps to reproduce

Step	1
Action	User logs in
Input data	Username: 'testuser' Password: 'testuser'
Expected result	

Step	2
Action	User navigates to 'test_budget'
Input data	
Expected result	

Step	3
Action	User enters income source
Input data	'Salary'
Expected result	

Step	4
Action	User enters income amount
Input data	3000
Expected result	

Step	5
Action	User clicks 'Add Income'
Input data	
Expected result	

Step	6
Action	User views the list of incomes for 'test_budget'
Input data	
Expected result	Incomes: Salary: $3000.0

Step	7
Action	User views the total income amount for 'test_budget'
Input data	
Expected result	Total Income: $3000.0

STP-54

Generate Budget Report

This test case verifies that a user can generate a comprehensive budget report, including all expenses and incomes, and view the summary.

Severity	**Priority**	**Type**	**Layer**
— Not set	— Not set	Acceptance	Unit
Milestone	**Automation**	**Status**	**Is Muted**
-	Manual	Actual	No

Pre-conditions

- The user has an existing budget named "test_budget".
- The budget contains some expenses and incomes.

Post-conditions

The user has access to a comprehensive budget report.

Steps to reproduce

Step	1
Action	The user logs into the application
Input data	Username: 'testuser' Password: 'testuser'
Expected result	

Step	2
Action	The user navigates to budget 'test_budget'
Input data	
Expected result	

Step	3
Action	User clicks on the 'Generate Report' button
Input data	
Expected result	

Step	4
Action	User views generated report, which includes all expenses and incomes
Input data	
Expected result	List of expenses and incomes with their categories, amounts, and totals Summary of total expenses, total incomes, net balance.

STP-55

User Manages Budget

This test case verifies the end-to-end functionality of the budget application, including creating a budget, adding expenses and incomes, generating a report, and deleting entries.

Severity	**Priority**	**Type**	**Layer**
— Not set	— Not set	Other	Not set
Milestone	**Automation**	**Status**	**Is Muted**
-	Manual	Actual	No

Pre-conditions

- The user has an account and is logged into the budget application.
- The database is connected, and the application is running.

Steps to reproduce

Step	1
Action	User logs into the budget application
Input data	Username: 'testuser' Password: 'testuser'
Expected result	

Step	2
Action	User enters budget name
Input data	'Monthly Budget'
Expected result	

Step	3
Action	User clicks 'Create Budget' button
Input data	
Expected result	

Step	4
Action	User verifies that the budget 'Monthly Budget' appears in the budget list on the dashboard
Input data	
Expected result	Your Budgets Monthly Budget

Step	5

Action	User enters an income source
Input data	'Salary'
Expected result	

Step	6
Action	User enters an income amount
Input data	3000
Expected result	

Step	7
Action	User clicks the 'Add Income' button
Input data	
Expected result	

Step	8
Action	User verifies that the income appears in the income list for 'Monthly Budget'
Input data	
Expected result	Incomes: Salary: $3000.0

Step	9
Action	User selects an expense category
Input data	'Groceries'
Expected result	

Step	10
Action	User enters an expense amount
Input data	500
Expected result	

Step	11
Action	User clicks the 'Add Expense' button
Input data	
Expected result	

Step	12
Action	User verifies that the expense appears in the expense list for 'Monthly Budget'
Input data	
Expected result	Expenses by Category: Groceries: $500.0

Step	13
Action	User clicks on the 'Generate Report' button for 'Monthly Budget';
Input data	
Expected result	

Step	14
Action	User verifies the report shows the correct total income, total expenses, and savings
Input data	
Expected result	Total Income: $3000.0 Total Expenses: $500.0 Savings: $2500.0

Step	15
Action	User selects the 'Delete' button for 'Groceries: $500.0'
Input data	
Expected result	

Step	16
Action	User verifies that the expense 'Groceries' is removed from the expense list
Input data	
Expected result	Expenses by Category

Step	17
Action	User selects the 'Delete' button for 'Salary: $3000'
Input data	
Expected result	

Step	18

Action	User verifies that the income 'Salary' is removed from the income list.
Input data	
Expected result	Incomes

Step	19
Action	User selects 'Delete Budget' button for the 'Monthly Budget'
Input data	
Expected result	

Step	20
Action	User verifies that the budget 'Monthly Budget' is removed from the budget list on the dashboard.
Input data	
Expected result	Your Budgets No budgets found.

STP-56

User Registration

This test case verifies that a new user can successfully register an account.

Severity	Priority	Type	Layer
— Not set	— Not set	Acceptance	Unit
Milestone	**Automation**	**Status**	**Is Muted**
-	Manual	Actual	No

Pre-conditions

The user is on the registration page of the application.

Post-conditions

- The new user account (testuser) is created and stored in the database.
- The user can log in using the newly created account credentials.
- The user is redirected to the login page after successful registration.

Steps to reproduce

Step	1
Action	User enters a username and password and
Input data	username: testuser password: testuser
Expected result	

Step	2
Action	User clicks the "Register" button
Input data	
Expected result	

Step	3
Action	Verify the user has been redirected to the login page
Input data	
Expected result	Welcome to the Budget App Log In

STP-57

User Login

This test case verifies that a registered user can successfully log in to the application.

Severity	Priority	Type	Layer
— Not set	— Not set	Acceptance	Unit
Milestone	**Automation**	**Status**	**Is Muted**
-	Manual	Actual	No

Pre-conditions

- The user has successfully registered an account (e.g., testuser with password testuser).
- The user is on the login page of the application

Post-conditions

- The user session is created and maintained.
- The user can access the dashboard and use the application's functionalities.
- The user's login status is reflected on the dashboard (e.g., username displayed).

Steps to reproduce

Step	1
Action	User enters their registered username and password
Input data	username: testuser password: testuser
Expected result	

Step	2
Action	User clicks on the 'Login' button
Input data	
Expected result	

Step	3
Action	User verifies that their username is displayed on the dashboard, indicating that they are logged in
Input data	
Expected result	Dashboard Welcome, testuser!

STP-67 / Database Testing / Test Auth

Register User

Severity	**Priority**	**Type**	**Layer**
○ Normal	— Not set	Other	Not set
Milestone	**Automation**	**Status**	**Is Muted**
-	Manual	Actual	No

STP-68 / Database Testing / Test Auth

Register Existing User

Severity	**Priority**	**Type**	**Layer**
○ Normal	— Not set	Other	Not set
Milestone	**Automation**	**Status**	**Is Muted**
-	Manual	Actual	No

STP-69 / Database Testing / Test Auth

Login User

Severity	**Priority**	**Type**	**Layer**
○ Normal	— Not set	Other	Not set
Milestone	**Automation**	**Status**	**Is Muted**
-	Manual	Actual	No

STP-70 / Database Testing / Test Auth

Login with Invalid Credentials

Severity	**Priority**	**Type**	**Layer**
○ Normal	— Not set	Other	Not set
Milestone	**Automation**	**Status**	**Is Muted**
-	Manual	Actual	No

STP-71 / Database Testing / Test Auth

Register Existing Username

Severity	**Priority**	**Type**	**Layer**
○ Normal	— Not set	Other	Not set
Milestone	**Automation**	**Status**	**Is Muted**
-	Manual	Actual	No

STP-58 / Database Testing / Test Data Operations

Insert Document

This tests whether or not a document can be successfully inserted into the collection.

Severity	Priority	Type	Layer
○ Normal	— Not set	Other	Not set
Milestone	**Automation**	**Status**	**Is Muted**
-	Manual	Actual	No

STP-59 / Database Testing / Test Data Operations

Find Document

This test is designed to retrieve a document from the database given a specific username that corresponds to a document already in the database.

Severity	Priority	Type	Layer
○ Normal	— Not set	Other	Not set
Milestone	**Automation**	**Status**	**Is Muted**
-	Manual	Actual	No

STP-60 / Database Testing / Test Data Operations

Update Document

Severity	**Priority**	**Type**	**Layer**
○ Normal	— Not set	Other	Not set
Milestone	**Automation**	**Status**	**Is Muted**
-	Manual	Actual	No

STP-61 / Database Testing / Test Data Operations

Delete Document

Severity	Priority	Type	Layer
○ Normal	— Not set	Other	Not set
Milestone	**Automation**	**Status**	**Is Muted**
-	Manual	Actual	No

STP-62 / Database Testing / Test Data Operations

Insert Multiple Documents

Severity	**Priority**	**Type**	**Layer**
○ Normal	— Not set	Other	Not set
Milestone	**Automation**	**Status**	**Is Muted**
-	Manual	Actual	No

STP-63 / Database Testing / Test Data Operations

Find Nonexistent Document

Severity	Priority	Type	Layer
○ Normal	— Not set	Other	Not set
Milestone	**Automation**	**Status**	**Is Muted**
-	Manual	Actual	No

STP-64 / Database Testing / Test Data Operations

Update Nonexistent Document

Severity	**Priority**	**Type**	**Layer**
○ Normal	— Not set	Other	Not set
Milestone	**Automation**	**Status**	**Is Muted**
-	Manual	Actual	No

STP-65 / Database Testing / Test Data Operations

Delete Nonexistent Document

Severity	Priority	Type	Layer
○ Normal	— Not set	Other	Not set

Milestone	Automation	Status	Is Muted
-	Manual	Actual	No

STP-66 / Database Testing / Test Data Operations

Find All Documents

Severity	Priority	Type	Layer
○ Normal	— Not set	Other	Not set
Milestone	**Automation**	**Status**	**Is Muted**
-	Manual	Actual	No