Playwright Test Scripts for ReactJS Apartment Accounting System

Based on the ReactJS code and user stories, I'll create comprehensive Playwright test scripts to cover the key functionality. The tests will target the application running at `http://localhost:8000`.

Test Setup

First, let's create a base test file with common utilities:

javascript

```javascript
// tests/utils/baseTest.js
const { test, expect } = require('@playwright/test');

// Helper function to login
async function login(page, username = 'accountant', password = 'password12
3') {
  await page.goto('http://localhost:8000/login');
  await page.fill('input[name="username"]', username);
  await page.fill('input[name="password"]', password);
  await page.click('button[type="submit"]');
  await expect(page).toHaveURL('http://localhost:8000/');
}

// Helper function to create account
async function createAccount(page, accountData) {
  await page.goto('http://localhost:8000/accounts');
  await page.click('button:has-text("Create New Account")');

  await page.fill('input[name="name"]', accountData.name);
  await page.selectOption('select[name="type"]', accountData.type);
  if (accountData.description) {
    await page.fill('input[name="description"]', accountData.description);
  }
  await page.fill('input[name="opening_balance"]', accountData.opening_bal
ance);

  await page.click('button:has-text("Create Account")');
  await expect(page.locator('.Toastify__toast--success')).toBeVisible();
}

module.exports = { login, createAccount };
```

Account Management Tests (US-001, US-002, US-003)

javascript

```javascript
// tests/accountManagement.spec.js
const { test, expect } = require('@playwright/test');
const { login, createAccount } = require('./utils/baseTest');
```

```
test.describe('Account Management', () => {
  test.beforeEach(async ({ page }) => {
    await login(page);
  });

  test('US-001: Create financial accounts', async ({ page }) => {
    const accountData = {
      name: 'Test Asset Account',
      type: 'ASSET',
      description: 'Test account description',
      opening_balance: '1000.00'
    };

    await createAccount(page, accountData);

    // Verify account appears in the list
    await expect(page.locator(`text=${accountData.name}`)).toBeVisible();
    await expect(page.locator(`text=$${accountData.opening_balance}`)).toB
eVisible();
  });

  test('US-002: Update financial accounts', async ({ page }) => {
    // First create an account to update
    const accountData = {
      name: 'Account to Update',
      type: 'LIABILITY',
      opening_balance: '500.00'
    };
    await createAccount(page, accountData);

    // Find and click the edit button for the account
    const accountCard = page.locator('.account-card', { hasText: accountDa
ta.name });
    await accountCard.locator('button:has-text("Edit")').click();

    // Update the account details
    const updatedName = 'Updated Account Name';
    await page.fill('input[name="name"]', updatedName);
    await page.selectOption('select[name="type"]', 'EXPENSE');
    await page.fill('input[name="opening_balance"]', '750.00');
    await page.click('button:has-text("Update Account")');

    // Verify updates
    await expect(page.locator('.Toastify__toast--success')).toBeVisible();
    await expect(page.locator(`text=${updatedName}`)).toBeVisible();
    await expect(page.locator('text=$750.00')).toBeVisible();
  });

  test('US-003: View accounts with filters', async ({ page }) => {
    // Create test accounts of different types
    const assetAccount = {
```

```
      name: 'Test Asset',
      type: 'ASSET',
      opening_balance: '1000.00'
    };
    const liabilityAccount = {
      name: 'Test Liability',
      type: 'LIABILITY',
      opening_balance: '500.00'
    };
    await createAccount(page, assetAccount);
    await createAccount(page, liabilityAccount);

    // Test filtering
    await page.selectOption('select[name="accountTypeFilter"]', 'ASSET');
    await expect(page.locator(`text=${assetAccount.name}`)).toBeVisible();
    await expect(page.locator(`text=${liabilityAccount.name}`)).not.toBeVi
sible();

    // Test sorting
    await page.selectOption('select[name="accountTypeFilter"]', 'ALL');
    await page.click('button:has-text("Sort by Balance")');
    const balances = await page.locator('.account-balance').allTextContent
s();
    const parsedBalances = balances.map(b => parseFloat(b.replace(/[^0-9.-
]+/g, '')));
    expect(parsedBalances).toEqual(parsedBalances.sort((a, b) => a - b));
  });
});
```

Transaction Management Tests (US-004, US-005)

javascript

```
// tests/transactionManagement.spec.js
const { test, expect } = require('@playwright/test');
const { login, createAccount } = require('./utils/baseTest');

test.describe('Transaction Management', () => {
  test.beforeEach(async ({ page }) => {
    await login(page);

    // Create accounts for transaction testing
    const bankAccount = {
      name: 'Bank Account',
      type: 'ASSET',
      opening_balance: '1000.00'
    };
    const expenseAccount = {
      name: 'Office Expenses',
      type: 'EXPENSE',
      opening_balance: '0.00'
    };
    await createAccount(page, bankAccount);
```

```
    await createAccount(page, expenseAccount);
  });

  test('US-004: Record transactions', async ({ page }) => {
    await page.goto('http://localhost:8000/transactions');
    await page.click('button:has-text("Record New Transaction")');

    // Fill transaction form
    await page.selectOption('select[name="account_id"]', { label: 'Bank Ac
count (ASSET)' });
    await page.selectOption('select[name="contra_account_id"]', { label: '
Office Expenses (EXPENSE)' });
    await page.fill('input[name="amount"]', '150.00');
    await page.fill('input[name="description"]', 'Office supplies');
    await page.click('button:has-text("Record Transaction")');

    // Verify success
    await expect(page.locator('.Toastify__toast--success')).toBeVisible();
    await expect(page.locator('text=Office supplies')).toBeVisible();
    await expect(page.locator('text=$150.00')).toBeVisible();

    // Verify account balances updated
    await page.goto('http://localhost:8000/accounts');
    await expect(page.locator('text=Bank Account').locator('..').locator('
.account-balance')).toHaveText('$850.00');
  });

  test('US-005: Prevent negative balances', async ({ page }) => {
    await page.goto('http://localhost:8000/transactions');
    await page.click('button:has-text("Record New Transaction")');

    // Attempt invalid transaction
    await page.selectOption('select[name="account_id"]', { label: 'Bank Ac
count (ASSET)' });
    await page.selectOption('select[name="contra_account_id"]', { label: '
Office Expenses (EXPENSE)' });
    await page.fill('input[name="amount"]', '1500.00');
    await page.click('button:has-text("Record Transaction")');

    // Verify error
    await expect(page.locator('.Toastify__toast--error')).toBeVisible();
    await expect(page.locator('.Toastify__toast--error')).toContainText('I
nsufficient funds');

    // Verify balance unchanged
    await page.goto('http://localhost:8000/accounts');
    await expect(page.locator('text=Bank Account').locator('..').locator('
.account-balance')).toHaveText('$1,000.00');
  });

  test('US-004: Edge cases - duplicate and future transactions', async ({
page }) => {
```

```
    // First create a valid transaction
    await page.goto('http://localhost:8000/transactions');
    await page.click('button:has-text("Record New Transaction")');
    await page.selectOption('select[name="account_id"]', { label: 'Bank Ac
count (ASSET)' });
    await page.selectOption('select[name="contra_account_id"]', { label: '
Office Expenses (EXPENSE)' });
    await page.fill('input[name="amount"]', '100.00');
    await page.fill('input[name="description"]', 'Duplicate test');
    await page.click('button:has-text("Record Transaction")');
    await expect(page.locator('.Toastify__toast--success')).toBeVisible();

    // Attempt duplicate
    await page.click('button:has-text("Record New Transaction")');
    await page.selectOption('select[name="account_id"]', { label: 'Bank Ac
count (ASSET)' });
    await page.selectOption('select[name="contra_account_id"]', { label: '
Office Expenses (EXPENSE)' });
    await page.fill('input[name="amount"]', '100.00');
    await page.fill('input[name="description"]', 'Duplicate test');
    await page.click('button:has-text("Record Transaction")');
    await expect(page.locator('.Toastify__toast--warning')).toContainText(
'possible duplicate');

    // Future-dated transaction
    await page.click('button:has-text("Record New Transaction")');
    const futureDate = new Date();
    futureDate.setDate(futureDate.getDate() + 7);
    const futureDateStr = futureDate.toISOString().split('T')[0];
    await page.selectOption('select[name="account_id"]', { label: 'Bank Ac
count (ASSET)' });
    await page.selectOption('select[name="contra_account_id"]', { label: '
Office Expenses (EXPENSE)' });
    await page.fill('input[name="amount"]', '50.00');
    await page.fill('input[name="transaction_date"]', futureDateStr);
    await page.click('button:has-text("Record Transaction")');
    await expect(page.locator('.Toastify__toast--success')).toBeVisible();
    await expect(page.locator('text=Pending')).toBeVisible();
  });
});
```

Reporting Tests (US-006, US-007)

javascript

```
// tests/reporting.spec.js
const { test, expect } = require('@playwright/test');
const { login, createAccount } = require('./utils/baseTest');

test.describe('Reporting', () => {
  test.beforeEach(async ({ page }) => {
    await login(page);
```

```
    // Setup test data
    const assetAccount = {
      name: 'Test Asset',
      type: 'ASSET',
      opening_balance: '2000.00'
    };
    const liabilityAccount = {
      name: 'Test Liability',
      type: 'LIABILITY',
      opening_balance: '500.00'
    };
    const incomeAccount = {
      name: 'Test Income',
      type: 'INCOME',
      opening_balance: '0.00'
    };
    const expenseAccount = {
      name: 'Test Expense',
      type: 'EXPENSE',
      opening_balance: '0.00'
    };
    await createAccount(page, assetAccount);
    await createAccount(page, liabilityAccount);
    await createAccount(page, incomeAccount);
    await createAccount(page, expenseAccount);

    // Create some transactions
    await page.goto('http://localhost:8000/transactions');
    await page.click('button:has-text("Record New Transaction")');
    await page.selectOption('select[name="account_id"]', { label: 'Test In
come (INCOME)' });
    await page.selectOption('select[name="contra_account_id"]', { label: '
Test Asset (ASSET)' });
    await page.fill('input[name="amount"]', '1000.00');
    await page.fill('input[name="description"]', 'Income transaction');
    await page.click('button:has-text("Record Transaction")');

    await page.click('button:has-text("Record New Transaction")');
    await page.selectOption('select[name="account_id"]', { label: 'Test As
set (ASSET)' });
    await page.selectOption('select[name="contra_account_id"]', { label: '
Test Expense (EXPENSE)' });
    await page.fill('input[name="amount"]', '500.00');
    await page.fill('input[name="description"]', 'Expense transaction');
    await page.click('button:has-text("Record Transaction")');
  });

  test('US-006: Generate account status report', async ({ page }) => {
    await page.goto('http://localhost:8000/reports');
    await page.click('button:has-text("Balance Report")');

    // Verify report content
```

```
    await expect(page.locator('text=Balance Sheet Report')).toBeVisible();
    await expect(page.locator('text=Test Asset')).toBeVisible();
    await expect(page.locator('text=$2,500.00')).toBeVisible(); // 2000 +
1000 - 500
    await expect(page.locator('text=Test Liability')).toBeVisible();
    await expect(page.locator('text=$500.00')).toBeVisible();
    await expect(page.locator('text=Net Worth')).toBeVisible();
    await expect(page.locator('text=$2,000.00')).toBeVisible(); // 2500 -
500
  });

  test('US-007: Monthly profit/loss report', async ({ page }) => {
    await page.goto('http://localhost:8000/reports');
    await page.click('button:has-text("Profit & Loss")');

    // Verify report content
    await expect(page.locator('text=Profit & Loss Statement')).toBeVisible
();
    await expect(page.locator('text=Total Income')).toBeVisible();
    await expect(page.locator('text=$1,000.00')).toBeVisible();
    await expect(page.locator('text=Total Expenses')).toBeVisible();
    await expect(page.locator('text=($500.00)')).toBeVisible();
    await expect(page.locator('text=Net Profit/Loss')).toBeVisible();
    await expect(page.locator('text=$500.00')).toBeVisible();
  });
});
```

Authentication Tests

javascript

```
// tests/auth.spec.js
const { test, expect } = require('@playwright/test');

test.describe('Authentication', () => {
  test('Successful login', async ({ page }) => {
    await page.goto('http://localhost:8000/login');
    await page.fill('input[name="username"]', 'accountant');
    await page.fill('input[name="password"]', 'password123');
    await page.click('button[type="submit"]');
    await expect(page).toHaveURL('http://localhost:8000/');
    await expect(page.locator('text=Welcome')).toBeVisible();
  });

  test('Failed login', async ({ page }) => {
    await page.goto('http://localhost:8000/login');
    await page.fill('input[name="username"]', 'wronguser');
    await page.fill('input[name="password"]', 'wrongpass');
    await page.click('button[type="submit"]');
    await expect(page.locator('.Toastify__toast--error')).toBeVisible();
    await expect(page.locator('.Toastify__toast--error')).toContainText('L
ogin failed');
  });
```

```
  test('Logout', async ({ page }) => {
    await page.goto('http://localhost:8000/login');
    await page.fill('input[name="username"]', 'accountant');
    await page.fill('input[name="password"]', 'password123');
    await page.click('button[type="submit"]');
    await page.click('text=Logout');
    await expect(page).toHaveURL('http://localhost:8000/login');
  });
});
```

Running the Tests

1. Install Playwright:

bash

```
npm init playwright@latest
```

2. Create a `playwright.config.js` file:

javascript

```
// playwright.config.js
module.exports = {
  testDir: './tests',
  timeout: 30000,
  expect: {
    timeout: 5000
  },
  fullyParallel: true,
  forbidOnly: !!process.env.CI,
  retries: process.env.CI ? 2 : 0,
  workers: process.env.CI ? 1 : undefined,
  reporter: 'html',
  use: {
    actionTimeout: 0,
    baseURL: 'http://localhost:8000',
    trace: 'on-first-retry',
  },
  projects: [
    {
      name: 'chromium',
      use: {
        channel: 'chromium',
      },
    },
    {
      name: 'firefox',
      use: {
        browserName: 'firefox',
      },
    },
    {
      name: 'webkit',
      use: {
        browserName: 'webkit',
```

```
      },
    },
  ],
};
```

3. Run the tests:

bash

```
npx playwright test
```

4. View the report:

bash

```
npx playwright show-report
```

These tests cover all the user stories with their acceptance criteria, including edge cases and error handling scenarios. The tests are designed to run against the React frontend at `http://localhost:8000` and verify both the UI behavior and the integration with the backend API.