Portfolio Tracker Pro

Modular Code Reference Guide

Total Functions	61
Total Modules	8
Original File Size	3,194 lines
Largest Module	ui-components.js (18 functions)
Generated	October 28, 2025

Table of Contents

1. Architecture Overview

The Portfolio Tracker Pro has been successfully modularized from a single 3,194-line file into 8 focused modules. This modular architecture improves maintainability, code organization, and development workflow.

Module Dependencies

```
The modules are loaded in dependency order: 1. globals.js - Shared variables and constants 2. calculations.js - Pure mathematical functions 3. data-manager.js - Data persistence and management 4. api-service.js - External services and APIs 5. ui-components.js - User interface interactions 6. dividends.js - Dividend-specific features 7. cash-flow.js - Cash flow management 8. main.js - Application initialization and coordination
```

2. Module Details

2.1 globals.js

Functions: 0

Purpose: Shared variables and constants

Contains all global variables, constants, and shared state that needs to be accessible across all modules.

Contents:

- CACHE_DURATION - API cache duration constant
- transactions - Global transactions array
- portfolios - Portfolio configuration array
- livePrices - Real-time price data object
- cashFlows - Cash flow data array
- globalSymbolData - Symbol information cache
- activeTickerFilter - Current ticker filter state
- originalSidebarHTML - Sidebar content backup
- sortState - Table sorting configurations
- transactionFilters - Filter state object
- PORTFOLIO_COLORS - Color scheme constants

2.2 calculations.js

Functions: 7

Purpose: Mathematical calculations and date operations

Pure calculation functions with no side effects. Handles all mathematical operations including XIRR calculations and date formatting.

Contents:

- formatDateDDMMYYYY() - Convert dates to DD/MM/YYYY format
- calculateDaysHeld() - Calculate days between two dates
- isValidTransaction() - Validate transaction data structure
- isValidForXIRR() - Validate data for XIRR calculation
- calculateXIRR() - Newton-Raphson XIRR calculation
- calculateXIRRForSymbol() - Symbol-specific XIRR calculation
- calculatePortfolioXIRR() - Portfolio-wide XIRR calculation

2.3 data-manager.js

Functions: 15

Purpose: Data persistence, portfolio management, transactions

Handles all data operations including localStorage, Supabase integration, transaction management, and portfolio operations.

Contents:

- populatePortfolioFilter() - Populate portfolio dropdown filters

- checkFirstVisit() - Show welcome modal for new users
- checkApiKey() - Validate API key presence
- saveApiKey() - Store API key to localStorage
- loadApiKey() - Retrieve API key from localStorage
- initializePortfolios() - Initialize portfolio system
- updatePortfolioDropdown() - Update portfolio selection dropdown
- updatePortfolioTabs() - Update portfolio tab navigation
- updatePortfolioList() - Update portfolio management list
- addPortfolio() - Create new portfolio
- updatePortfolioName() - Rename existing portfolio
- savePortfolios() - Persist portfolios to storage
- getPortfolioName() - Get portfolio name by ID
- getPortfolioColorDot() - Get portfolio color indicator
- Plus: loadDataFromSupabase(), saveDataToSupabase(), fixBadDates(), addTransaction(), confirmClearData(), confirmDeleteSelected(), deletePortfolio()

2.4 api-service.js

Functions: 13

Purpose: API calls, price fetching, CSV operations

Manages external API communications, price fetching from Twelve Data API, CSV import/export operations, and manual price management.

Contents:

- checkDuplicateTransaction() - Detect duplicate transactions
- handleCsvImport() - Handle CSV file import
- exportTransactionsToCSV() - Export data to CSV
- downloadCsvTemplate() - Generate CSV template
- searchTicker() - Search for ticker symbols
- clearTickerSearch() - Clear ticker search results
- refreshPricesAndNames() - Refresh UI with current data
- initializePriceMode() - Initialize manual/API price mode
- showSaveButton() - Show manual price save button
- removeSaveButton() - Remove manual price save button
- saveManualPrices() - Save manually entered prices
- getCurrentPrice() - Get current price (manual or API)
- handlePriceCheckbox() - Handle price editing checkbox
- Plus: processCsvData(), savePricesCache(), fetchLivePrices(), getLivePrice(), refreshAllPrices()

2.5 ui-components.js

Functions: 18

Purpose: User interface interactions, modals, table operations

Handles all user interface interactions including modals, table operations, sorting, filtering, and form management.

Contents:

- debounce() - Debounce user input
- makeTickerClickable() - Make ticker symbols clickable
- filterByTicker() - Filter view by ticker symbol
- showTickerModal() - Display ticker detail modal
- closeTickerModal() - Close ticker modal
- editTransactionFromModal() - Trigger edit from ticker modal
- convertToDateInputFormat() - Convert date for input fields
- showEditModal() - Display transaction edit modal
- closeEditModal() - Close edit modal
- formatDateInput() - Smart date input formatting
- applyTransactionFilters() - Apply table filters
- clearTransactionFilters() - Clear all filters
- initializeTabs() - Initialize tab navigation system
- initializeSortListeners() - Setup table sorting
- sortTable() - Sort table by column
- convertDDMMYYYYtoYYYYMMDD() - Convert date formats
- updateCsvHelpModal() - Update CSV help content
- generatePriceCell() - Generate price table cell
- Plus: saveEditedTransaction()

2.6 dividends.js

Functions: 4

Purpose: Dividend tracking and management

Specialized module for dividend tracking, DRIP vs cash dividend handling, and dividend-specific UI components.

Contents:

- updateDividendsTable() - Update dividends display table
- updateDividendsSidebar() - Update dividend sidebar content
- handleDividendTypeChange() - Handle DRIP vs Cash dividend toggle
- initializePriceMode() - Initialize price mode (duplicate function)

2.7 cash-flow.js

Functions: 2

Purpose: Cash flow management and analysis

Handles cash flow tracking, analysis, and XIRR calculations specific to cash flow data.

Contents:

- calculateCashFlowXIRR() - Calculate XIRR for cash flows
- updateCashFlowTable() - Update cash flow display table
- Plus: handleCashFlowCsvImport(), addCashFlow(), deleteCashFlowSelected()

2.8 main.js

Functions: 2

Purpose: Application coordination and initialization

Central coordination module that initializes the application, sets up event listeners, and coordinates between all other modules.

Contents:

- updateTables() - Master function to update all tables
- updateSummary() - Update summary statistics
- init() - Main initialization function with all event listeners
- Contains all event listener setup and application startup logic

3. Function Reference

Task	Module	Key Functions
Mathematical Calculations	calculations.js	calculateXIRR(), formatDateDDMMYYYY()
Data Storage & Retrieval	data-manager.js	saveDataToSupabase(), loadDataFromSupabase()
Portfolio Management	data-manager.js	addPortfolio(), updatePortfolioName()
Transaction Management	data-manager.js	addTransaction(), confirmDeleteSelected()
Price Fetching	api-service.js	fetchLivePrices(), refreshAllPrices()
CSV Operations	api-service.js	handleCsvImport(), exportTransactionsToCSV()
User Interface	ui-components.js	showTickerModal(), initializeTabs()
Table Operations	ui-components.js	sortTable(), applyTransactionFilters()
Dividend Tracking	dividends.js	updateDividendsTable(), handleDividendTypeChange()
Cash Flow Analysis	cash-flow.js	calculateCashFlowXIRR(), updateCashFlowTable()
Application Startup	main.js	init(), updateTables()

4. Quick Reference Guide

HTML Script Loading Order

Development Guidelines

- Add new mathematical functions to calculations.js
- Add new data operations to data-manager.js
- Add new API integrations to api-service.js
- Add new UI components to ui-components.js
- Add dividend features to dividends.js
- Add cash flow features to cash-flow.js
- Add initialization code to main.js
- Add shared variables to globals.js

Benefits of Modular Structure

- Improved code maintainability and readability
- Easier debugging and testing of individual components
- Better code organization and separation of concerns
- Simplified collaboration among multiple developers
- Reduced risk of conflicts when making changes
- Enhanced code reusability across different projects
- Clearer understanding of application architecture