RAGA NEOBANK SDK

Technical Documentation for Stakeholders

Core SDK | React Hooks | API Reference

Version 0.1.0

Executive Summary

The Raga Neobank SDK is a developer toolkit that enables applications to interact with the Raga Finance infrastructure. It is split into two packages: a universal Core SDK (works anywhere JavaScript runs) and a React Hooks package (purpose-built for React/Next.js frontends).

This document provides a non-technical overview of every function available, the data each function returns, and how the packages relate to one another.

Package Overview

	@raga-neobank/core	@raga-neobank/react
Purpose	Universal API client	React hooks & context
Runtime	Node.js, Browser, CLI	React 19+ / Next.js
Dependencies	Zero (uses native fetch)	Core SDK + TanStack Query
Authentication	API Key + Wallet Address	Inherited from Core
Version	0.1.0	0.1.0

Core SDK: @raga-neobank/core

The Core SDK is the foundation layer. It communicates directly with the Raga Finance backend API and returns strongly-typed data. Every React hook internally delegates to Core SDK methods.

Vaults Module

Public endpoints to browse available investment vaults. No authentication required.

sdk.vaults.list()

What it does: Fetches all available vaults from the platform.

Authentication: Not required (public endpoint).

Returns: An array of Vault objects.

Vault Object Fields:

Field	Type	Description
`id`	string (UUID)	Unique vault identifier

`curatorId`	string	ID of the vault curator/manager
`vaultName`	string	Human-readable vault name
`vaultAddress`	string	On-chain contract address (0x…)
`chainId`	number	Blockchain network ID (e.g. 8453 = Base)
`isEnabled`	boolean	Whether the vault is currently active
`depositEnabled`	boolean	Whether new deposits are accepted
`strategyAllocations`	array	List of strategies with allocation percentages

sdk.vaults.get(vaultId)

What it does: Fetches details for a single vault by its UUID.

Returns: A single Vault object (same structure as above).

User Module

Authenticated endpoints for user account management.

sdk.user.getUser()

What it does: Fetches the authenticated user's profile details.

Authentication: Required (API key + wallet address).

Returns: A User object.

User Object Fields:

Field	Type	Description
`id`	string	Unique user identifier
`address`	string	User's Ethereum wallet address
`isEnabled`	boolean	Whether the account is active
`bankId`	string	Associated bank/institution ID
`createdOn`	string (ISO date)	Account creation timestamp
`updatedOn`	string (ISO date)	Last profile update timestamp

Portfolio Module

Authenticated endpoints for tracking investment positions.

sdk.portfolio.getPortfolio()

What it does: Fetches the user's complete portfolio including all vault positions, deposit values, and current values.

Authentication: Required.

Returns: A Portfolio object containing bank info and an array of positions.

Portfolio Object Fields:

Field	Type	Description
`bank.name`	string	Name of the managing bank
`bank.legalName`	string	Legal entity name of the bank
`walletAddress`	string	User's wallet address
`positions`	array	Array of PortfolioPosition objects (see below)

PortfolioPosition Fields:

Field	Type	Description
`vaultName`	string	Name of the vault
`vaultAddress`	string	On-chain vault contract address
`chainId`	number	Blockchain network identifier
`depositValueInAsset`	string	Original deposit amount in token units
`depositValueInUsd`	string	Original deposit value in USD
`currentValueInAsset`	string	Current value in token units
`currentValueInUsd`	string	Current value in USD
`decimals`	number	Token decimal precision (e.g. 6 for USDC)

Transactions Module

Authenticated endpoints for generating blockchain transaction payloads. These do not execute transactions directly – they return the data your application needs to submit transactions to the blockchain.

sdk.transactions.buildDepositPayload(request)

What it does: Generates a simulation-verified deposit transaction payload.

sdk.transactions.buildWithdrawPayload(request)

What it does: Generates a simulation-verified withdrawal transaction payload.

sdk.transactions.buildRedeemPayload(request)

What it does: Generates a simulation-verified redemption transaction payload.

All three methods accept the same request and return the same response structure:

Request Parameters:

Parameter	Type	Description
`vaultId`	string (UUID)	Target vault identifier
`amount`	string	Amount in smallest units (e.g. "1000000" = 1 USDC)
`chainId`	number	Blockchain network ID

Response – TransactionPayload:

Field	Type	Description
`vaultId`	string	Target vault UUID
`vaultName`	string	Human-readable vault name
`chainId`	number	Target blockchain network
`txs`	array	Ordered list of transaction steps to execute
`summary`	object	Human-readable transaction summary

TransactionStep Fields:

Field	Type	Description
`step`	number	Step order (1, 2, 3…)
`type`	string	approve | deposit | withdraw | redeem
`description`	string	Human-readable step description
`to / from`	string	Contract addresses for the transaction
`data`	string	Encoded transaction calldata
`gasEstimate`	string	Estimated gas units
`simulationSuccess`	boolean	Whether simulation passed

React Package: @raga-neobank/react

The React package wraps every Core SDK method in a TanStack Query hook, providing automatic caching, loading/error states, background re-fetching, and React-friendly APIs. It is designed for use with Next.js App Router and React 19+.

Setup: NeobankProvider

What it does: Initializes the SDK and provides it to all child components via React Context. Also sets up TanStack Query.

Required props: config (apiKey, userAddress, baseUrl, timeout) and optionally a custom QueryClient.

Available Hooks

Hook	Type	Core Method	Returns
`useVaults()`	Query	sdk.vaults.list()	Vault[]
`useVault(id)`	Query	sdk.vaults.get(id)	Vault
`useUser()`	Query	sdk.user.getUser()	User
`usePortfolio()`	Query	sdk.portfolio.getPortfolio()	Portfolio
`useBuildDeposit()`	Mutation	sdk.transactions.buildDepositPayload()	TransactionPayload
`useBuildWithdraw()`	Mutation	sdk.transactions.buildWithdrawPayload()	TransactionPayload
`useBuildRedeem()`	Mutation	sdk.transactions.buildRedeemPayload()	TransactionPayload
`useNeobank()`	Context	–	{ sdk, config }

Query vs Mutation

- **Query hooks** (useVaults, useUser, usePortfolio) automatically fetch data on mount, cache results, and provide isLoading / error / data states.
- **Mutation hooks** (useBuildDeposit, useBuildWithdraw, useBuildRedeem) are triggered on-demand via a mutate() function and provide isPending / error / data states.

Query Keys (for cache management)

The package exports a neobankKeys factory that allows manual cache invalidation:

Key Factory	Scope
`neobankKeys.vaults.all()`	All vault list queries
`neobankKeys.vaults.detail(id)`	Single vault query by ID
`neobankKeys.user(address)`	User profile for a wallet
`neobankKeys.portfolio(address)`	Portfolio for a wallet

Error Handling

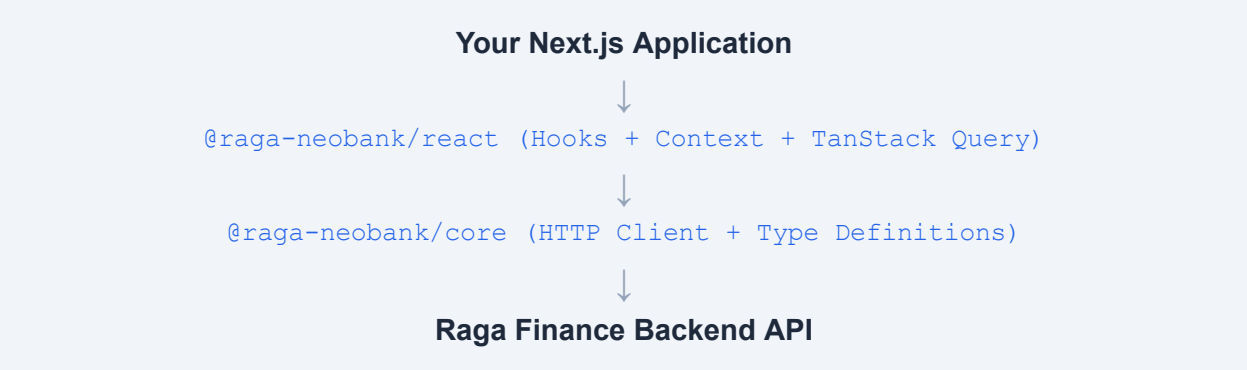
All SDK errors are thrown as NeobankError instances with structured fields:

Field	Type	Description
`message`	string	Human-readable error description
`code`	number	API error code
`statusCode`	number	HTTP status code
`detail`	string | null	Additional error context from the API

A type guard isNeobankError(error) is provided to safely check error instances.

Architecture Diagram

The relationship between the packages follows a clean layered architecture:



Summary

The Raga Neobank SDK provides a complete, type-safe interface to the Raga Finance platform. The Core package handles all API communication and can be used in any JavaScript environment, while the React package adds developer-friendly hooks with built-in state management for frontend applications. Together they enable rapid development of financial applications on the Raga platform.