@fin.cx/opendata

Documentation for @fin.cx/opendata

- readme.md for @fin.cx/opendata
- changelog.md for @fin.cx/opendata

readme.md for @fin.cx/opendata

□□ Complete financial intelligence toolkit for TypeScript

Access real-time stock prices, fundamental financial data, and comprehensive German company information - all through a single, unified API.

Installation

```
npm install @fin.cx/opendata
# or
pnpm add @fin.cx/opendata
```

Quick Start

□ Unified Stock Data API (Recommended)

Get complete stock data with automatic enrichment - the elegant way:

```
import { StockDataService, SecEdgarProvider } from '@fin.cx/opendata';

// Initialize unified service
const stockData = new StockDataService();

// Register providers
stockData.registerFundamentalsProvider(new SecEdgarProvider({
  userAgent: 'YourCompany youremail@example.com'
}));

// □ Get complete stock data with ONE method call
```

```
const apple = await stockData.getStockData('AAPL');

console.log({
  company: apple.fundamentals.companyName,      // "Apple Inc."
  price: apple.price.price,                     // $270.37
  marketCap: apple.fundamentals.marketCap,      // $4.13T (auto-calculated!)
  peRatio: apple.fundamentals.priceToEarnings,  // 28.42 (auto-calculated!)
  pbRatio: apple.fundamentals.priceToBook,      // 45.12 (auto-calculated!)
  eps: apple.fundamentals.earningsPerShareDiluted,  // $6.13
  revenue: apple.fundamentals.revenue            // $385.6B
});

// Batch fetch with automatic enrichment
const stocks = await stockData.getBatchStockData(['AAPL', 'MSFT', 'GOOGL']);

stocks.forEach(stock => {
  console.log(`${stock.ticker}: $${stock.price.price.toFixed(2)}, ` +
              `P/E ${stock.fundamentals?.priceToEarnings?.toFixed(2)}`);
});
// Output:
// AAPL: $270.37, P/E 28.42
// MSFT: $425.50, P/E 34.21
// GOOGL: $142.15, P/E 25.63
```

Why use the unified API?

- **Single service** for both prices and fundamentals
- **Automatic enrichment** - Market cap, P/E, P/B calculated automatically
- **One method call** - No manual `enrichWithPrice()` calls
- **Simplified code** - Less boilerplate, more readable
- **Type-safe** - Full TypeScript support

Price-Only Data (Alternative)

If you only need prices without fundamentals:

```
import { StockDataService, MarketstackProvider } from '@fin.cx/opendata';

const stockData = new StockDataService();
stockData.registerPriceProvider(new MarketstackProvider('YOUR_API_KEY'));
```

```
// Get just the price
const price = await stockData.getPrice('AAPL');
console.log(`${price.ticker}: $${price.price}`);

// Or use StockPriceService directly for more control
import { StockPriceService } from '@fin.cx/opendata';

const stockService = new StockPriceService({ ttl: 60000 });
stockService.register(new MarketstackProvider('YOUR_API_KEY'));

const apple = await stockService.getData({ type: 'current', ticker: 'AAPL' });
console.log(`${apple.companyFullName}: $${apple.price}`);
```

□□ Cryptocurrency Prices

Get real-time crypto prices with the CoinGecko provider:

```
import { StockPriceService, CoinGeckoProvider } from '@fin.cx/opendata';

const stockService = new StockPriceService({ ttl: 30000 });

// Optional: Pass API key for higher rate limits
// const provider = new CoinGeckoProvider('your-api-key');
const provider = new CoinGeckoProvider();

stockService.register(provider);

// Fetch single crypto price (using ticker symbol)
const btc = await stockService.getPrice({ ticker: 'BTC' });
console.log(`${btc.ticker}: $${btc.price.toLocaleString()}`);
console.log(`24h Change: ${btc.changePercent.toFixed(2)}%`);
console.log(`24h Volume: $${btc.volume?.toLocaleString()}`);

// Or use CoinGecko ID directly
const ethereum = await stockService.getPrice({ ticker: 'ethereum' });

// Batch fetch multiple cryptos
const cryptos = await stockService.getPrices({
```

```
  tickers: ['BTC', 'ETH', 'USDT', 'BNB', 'SOL']
});

cryptos.forEach(crypto => {
  console.log(`${crypto.ticker}: $${crypto.price.toFixed(2)} (${crypto.changePercent >= 0 ?
'+' : ''}${crypto.changePercent.toFixed(2)}%)`);
});

// Historical crypto prices
const history = await stockService.getData({
  type: 'historical',
  ticker: 'BTC',
  from: new Date('2025-01-01'),
  to: new Date('2025-01-31')
});

// Intraday crypto prices (hourly)
const intraday = await stockService.getData({
  type: 'intraday',
  ticker: 'ETH',
  interval: '1hour',
  limit: 24 // Last 24 hours
});
```

□□ Fundamentals-Only Data (Alternative)

If you only need fundamentals without prices:

```
import { StockDataService, SecEdgarProvider } from '@fin.cx/opendata';

const stockData = new StockDataService();
stockData.registerFundamentalsProvider(new SecEdgarProvider({
  userAgent: 'YourCompany youremail@example.com'
}));

// Get just fundamentals
const fundamentals = await stockData.getFundamentals('AAPL');

console.log({
```

```
  company: fundamentals.companyName,
  eps: fundamentals.earningsPerShareDiluted,
  revenue: fundamentals.revenue,
  sharesOutstanding: fundamentals.sharesOutstanding
});

// Or use FundamentalsService directly
import { FundamentalsService } from '@fin.cx/opendata';

const fundamentalsService = new FundamentalsService();
fundamentalsService.register(new SecEdgarProvider({
  userAgent: 'YourCompany youremail@example.com'
}));

const data = await fundamentalsService.getFundamentals('AAPL');
```

German Business Intelligence

Access comprehensive German company data:

```
import { OpenData } from '@fin.cx/opendata';
import * as path from 'path';

// Configure directory paths
const openData = new OpenData({
  nogitDir: path.join(process.cwd(), '.nogit'),
  downloadDir: path.join(process.cwd(), '.nogit', 'downloads'),
  germanBusinessDataDir: path.join(process.cwd(), '.nogit', 'germanbusinessdata')
});
await openData.start();

// Search for companies
const results = await openData.handelsregister.searchCompany("Siemens AG");

// Get detailed information with documents
const details = await openData.handelsregister.getSpecificCompany({
  court: "Munich",
  type: "HRB",
```

```
  number: "6684"
});
```

Features

□□ Stock & Crypto Market Module

- **Real-Time Prices** - Live and EOD prices from Marketstack and CoinGecko
- **Cryptocurrency Support** - 13M+ crypto tokens with 24/7 market data via CoinGecko
- **Company Names** - Automatic company name extraction (e.g., "Apple Inc (NASDAQ:AAPL)")
- **Historical Data** - Up to 15 years of daily EOD prices with pagination
- **OHLCV Data** - Open, High, Low, Close, Volume for technical analysis
- **Exchange Filtering** - Query specific exchanges via MIC codes (XNAS, XLON, XNYS)
- **Smart Caching** - Data-type aware TTL (historical cached forever, EOD 24h, live 30s)
- **Batch Operations** - Fetch 100+ symbols in one request
- **Type-Safe API** - Full TypeScript support with discriminated unions
- **Multi-Provider** - Automatic fallback between providers

□□ Fundamental Data Module

- **SEC EDGAR Integration** - FREE fundamental data directly from SEC filings
- **Comprehensive Metrics** - EPS, Revenue, Assets, Liabilities, Cash Flow, and more
- **All US Public Companies** - Complete coverage of SEC-registered companies
- **Historical Filings** - Data back to ~2009
- **CIK Lookup** - Automatic ticker-to-CIK mapping with smart caching
- **Calculated Ratios** - Market Cap, P/E, P/B ratios when combined with prices
- **No API Key Required** - Direct access to SEC's public API
- **Rate Limit Management** - Built-in 10 req/sec rate limiting

□□□ German Business Intelligence

- **MongoDB Integration** - Scalable data storage for millions of records
- **Bulk JSONL Import** - Process multi-GB datasets efficiently
- **Handelsregister Automation** - Automated document retrieval
- **CRUD Operations** - Full database management with validation
- **Streaming Processing** - Handle large datasets without memory issues

Advanced Examples

Combined Market Analysis (Unified API)

Analyze multiple companies with automatic enrichment:

```
import { StockDataService, MarketstackProvider, SecEdgarProvider } from '@fin.cx/opendata';

// Setup unified service
const stockData = new StockDataService();
stockData.registerPriceProvider(new MarketstackProvider('YOUR_API_KEY'));
stockData.registerFundamentalsProvider(new SecEdgarProvider({
  userAgent: 'YourCompany youremail@example.com'
}));

// Analyze multiple companies with ONE call
const tickers = ['AAPL', 'MSFT', 'GOOGL', 'AMZN'];
const stocks = await stockData.getBatchStockData(tickers);

// All metrics are automatically calculated!
stocks.forEach(stock => {
  if (stock.fundamentals) {
    console.log(`\n${stock.fundamentals.companyName} (${stock.ticker})`);
    console.log(`  Price: $${stock.price.price.toFixed(2)}`);
    console.log(`  Market Cap: $${(stock.fundamentals.marketCap! / 1e9).toFixed(2)}B`);
    console.log(`  P/E Ratio: ${stock.fundamentals.priceToEarnings!.toFixed(2)}`);
    console.log(`  Revenue: $${(stock.fundamentals.revenue! / 1e9).toFixed(2)}B`);
    console.log(`  EPS: $${stock.fundamentals.earningsPerShareDiluted!.toFixed(2)}`);
  }
});

// Or analyze one-by-one with automatic enrichment
for (const ticker of tickers) {
  const stock = await stockData.getStockData(ticker);

  // Everything is already enriched - no manual calculations needed!
  console.log(`${ticker}: P/E ${stock.fundamentals?.priceToEarnings?.toFixed(2)}`);
```

```
}
```

Fundamental Data Screening

Screen stocks by financial metrics:

```
// Fetch data for multiple tickers
const tickers = ['AAPL', 'MSFT', 'GOOGL', 'AMZN', 'META', 'NVDA', 'TSLA'];
const stocks = await stockData.getBatchStockData(tickers);

// Filter by criteria (all metrics auto-calculated!)
const valueStocks = stocks.filter(stock => {
  const f = stock.fundamentals;
  return f &&
    f.priceToEarnings! < 30 &&          // P/E under 30
    f.priceToBook! < 10 &&              // P/B under 10
    f.revenue! > 100_000_000_000;       // Revenue > $100B
});

console.log('\n□□Value Stocks:');
valueStocks.forEach(stock => {
  console.log(`${stock.ticker}: P/E ${stock.fundamentals!.priceToEarnings!.toFixed(2)}, ` +
              `P/B ${stock.fundamentals!.priceToBook!.toFixed(2)}`);
});
```

Historical Data Analysis

Fetch and analyze historical price trends:

```
// Get 1 year of historical data
const history = await stockService.getData({
  type: 'historical',
  ticker: 'AAPL',
  from: new Date('2024-01-01'),
  to: new Date('2024-12-31'),
  sort: 'DESC' // Newest first
});
```

```
// Calculate statistics
const prices = history.map(p => p.price);
const high52Week = Math.max(...prices);
const low52Week = Math.min(...prices);
const avgPrice = prices.reduce((a, b) => a + b) / prices.length;

console.log(`52-Week Analysis for AAPL:`);
console.log(`  High:    $${high52Week.toFixed(2)}`);
console.log(`  Low:     $${low52Week.toFixed(2)}`);
console.log(`  Average: $${avgPrice.toFixed(2)}`);
console.log(`  Days:    ${history.length}`);

// Calculate Simple Moving Average
const calculateSMA = (data: IStockPrice[], period: number) => {
  const sma: number[] = [];
  for (let i = period - 1; i < data.length; i++) {
    const sum = data.slice(i - period + 1, i + 1)
      .reduce((acc, p) => acc + p.price, 0);
    sma.push(sum / period);
  }
  return sma;
};

const sma20 = calculateSMA(history, 20);
const sma50 = calculateSMA(history, 50);

console.log(`\nMoving Averages:`);
console.log(`  20-day SMA: $${sma20[0].toFixed(2)}`);
console.log(`  50-day SMA: $${sma50[0].toFixed(2)}`);
```

OHLCV Technical Analysis

Use OHLCV data for technical indicators:

```
const history = await stockService.getData({
  type: 'historical',
  ticker: 'TSLA',
  from: new Date('2024-11-01'),
  to: new Date('2024-11-30')
```

```
});

// Calculate daily trading range
for (const day of history) {
  const range = day.high! - day.low!;
  const rangePercent = (range / day.low!) * 100;

  console.log(`${day.timestamp.toISOString().split('T')[0]}:`);
  console.log(`  Open:   $${day.open}`);
  console.log(`  High:   $${day.high}`);
  console.log(`  Low:    $${day.low}`);
  console.log(`  Close:  $${day.price}`);
  console.log(`  Volume: ${day.volume?.toLocaleString()}`);
  console.log(`  Range:  $${range.toFixed(2)} (${rangePercent.toFixed(2)}%)`);
}
```

Fundamental Data Screening

Screen stocks based on fundamental metrics:

```
const tickers = ['AAPL', 'MSFT', 'GOOGL', 'AMZN', 'TSLA', 'META', 'NVDA'];

// Fetch fundamentals for all tickers
const allFundamentals = await fundamentalsService.getBatchFundamentals(tickers);

// Get current prices
const prices = await stockService.getData({
  type: 'batch',
  tickers: tickers
});

// Create price map
const priceMap = new Map(prices.map(p => [p.ticker, p.price]));

// Enrich with prices
const enriched = await fundamentalsService.enrichBatchWithPrices(
  allFundamentals,
  priceMap
);
```

```
// Screen for value stocks (P/E < 25, P/B < 5)
const valueStocks = enriched.filter(f =>
  f.priceToEarnings && f.priceToEarnings < 25 &&
  f.priceToBook && f.priceToBook < 5
);

console.log('Value Stocks:');
valueStocks.forEach(stock => {
  console.log(`\n${stock.companyName} (${stock.ticker})`);
  console.log(`  P/E Ratio: ${stock.priceToEarnings!.toFixed(2)}`);
  console.log(`  P/B Ratio: ${stock.priceToBook!.toFixed(2)}`);
  console.log(`  Market Cap: $${(stock.marketCap! / 1e9).toFixed(2)}B`);
});
```

Market Dashboard

Create a comprehensive market overview:

```
const indicators = [
  { ticker: 'AAPL', name: 'Apple' },
  { ticker: 'MSFT', name: 'Microsoft' },
  { ticker: 'GOOGL', name: 'Alphabet' },
  { ticker: 'AMZN', name: 'Amazon' },
  { ticker: 'TSLA', name: 'Tesla' }
];

const prices = await stockService.getData({
  type: 'batch',
  tickers: indicators.map(i => i.ticker)
});

// Display with color-coded changes
prices.forEach(price => {
  const indicator = indicators.find(i => i.ticker === price.ticker);
  const arrow = price.change >= 0 ? '↑' : '↓';
  const color = price.change >= 0 ? '\x1b[32m' : '\x1b[31m';

  console.log(
```

```
    `${price.companyName!.padEnd(25)} $${price.price.toFixed(2).padStart(8)} ` +
    `${color}${arrow} ${price.changePercent.toFixed(2)}%\x1b[0m`
  );
});
```

Exchange-Specific Trading

Compare prices across different exchanges:

```
// Vodafone trades on both London and NYSE
const exchanges = [
  { mic: 'XLON', name: 'London Stock Exchange' },
  { mic: 'XNYS', name: 'New York Stock Exchange' }
];

for (const exchange of exchanges) {
  try {
    const price = await stockService.getData({
      type: 'current',
      ticker: 'VOD',
      exchange: exchange.mic
    });

    console.log(`${exchange.name}:`);
    console.log(`  Price: ${price.price} ${price.currency}`);
    console.log(`  Volume: ${price.volume?.toLocaleString()}`);
    console.log(`  Exchange: ${price.exchangeName}`);
  } catch (error) {
    console.log(`${exchange.name}: Not available`);
  }
}
```

Configuration

Stock Service Options

```
const stockService = new StockPriceService({
  ttl: 60000,         // Default cache TTL in ms
  maxEntries: 10000   // Max cached entries
});

// Marketstack - EOD data (requires API key)
stockService.register(new MarketstackProvider('YOUR_API_KEY'), {
  enabled: true,
  priority: 100,
  timeout: 10000,
  retryAttempts: 3,
  retryDelay: 1000
});
```

Fundamentals Service Options

```
const fundamentalsService = new FundamentalsService({
  ttl: 90 * 24 * 60 * 60 * 1000,  // 90 days (quarterly refresh)
  maxEntries: 10000
});

// SEC EDGAR provider (FREE - no API key!)
fundamentalsService.register(new SecEdgarProvider({
  userAgent: 'YourCompany youremail@example.com',
  cikCacheTTL: 30 * 24 * 60 * 60 * 1000,      // 30 days
  fundamentalsCacheTTL: 90 * 24 * 60 * 60 * 1000,  // 90 days
  timeout: 30000
}));
```

Directory Configuration (German Business Data)

All directory paths are mandatory when using German business data features:

```
import { OpenData } from '@fin.cx/opendata';
import * as path from 'path';
```

```
// Development
const openData = new OpenData({
  nogitDir: path.join(process.cwd(), '.nogit'),
  downloadDir: path.join(process.cwd(), '.nogit', 'downloads'),
  germanBusinessDataDir: path.join(process.cwd(), '.nogit', 'germanbusinessdata')
});

// Production
const openDataProd = new OpenData({
  nogitDir: '/var/lib/myapp/data',
  downloadDir: '/var/lib/myapp/data/downloads',
  germanBusinessDataDir: '/var/lib/myapp/data/germanbusinessdata'
});
```

Environment Variables

Set environment variables for API keys and database:

```
# Marketstack API (for EOD stock data)
MARKETSTACK_COM_TOKEN=your_api_key_here

# MongoDB (for German business data)
MONGODB_URL=mongodb://localhost:27017
MONGODB_NAME=opendata
MONGODB_USER=myuser
MONGODB_PASS=mypass
```

API Reference

Stock Price Interfaces

```
interface IStockPrice {
  ticker: string;
  price: number;
  currency: string;
```

```
  change: number;
  changePercent: number;
  previousClose: number;
  timestamp: Date;
  provider: string;
  marketState: 'PRE' | 'REGULAR' | 'POST' | 'CLOSED';
  exchange?: string;
  exchangeName?: string;

  // OHLCV data
  volume?: number;
  open?: number;
  high?: number;
  low?: number;
  adjusted?: boolean;
  dataType: 'eod' | 'intraday' | 'live';
  fetchedAt: Date;

  // Company identification
  companyName?: string;       // "Apple Inc"
  companyFullName?: string;   // "Apple Inc (NASDAQ:AAPL)"
}
```

Fundamental Data Interfaces

```
interface IStockFundamentals {
  ticker: string;
  cik: string;
  companyName: string;
  provider: string;
  timestamp: Date;
  fetchedAt: Date;

  // Per-share metrics
  earningsPerShareBasic?: number;
  earningsPerShareDiluted?: number;
  sharesOutstanding?: number;
```

```
  // Income statement (annual USD)
  revenue?: number;
  netIncome?: number;
  operatingIncome?: number;
  grossProfit?: number;

  // Balance sheet (annual USD)
  assets?: number;
  liabilities?: number;
  stockholdersEquity?: number;
  cash?: number;
  propertyPlantEquipment?: number;

  // Calculated metrics (requires price)
  marketCap?: number;        // price × sharesOutstanding
  priceToEarnings?: number;  // price / EPS
  priceToBook?: number;      // marketCap / stockholdersEquity

  // Metadata
  fiscalYear?: string;
  fiscalQuarter?: string;
  filingDate?: Date;
  form?: '10-K' | '10-Q' | string;
}
```

Key Methods

StockPriceService

- `getData(request)` - Unified method for all stock data (current, historical, batch)
- `getPrice(request)` - Convenience method for single current price
- `getPrices(request)` - Convenience method for batch current prices
- `register(provider, config)` - Add data provider with priority and retry config
- `checkProvidersHealth()` - Test all providers and return health status
- `getProviderStats()` - Get success/error statistics for each provider
- `clearCache()` - Clear price cache
- `setCacheTTL(ttl)` - Update cache TTL dynamically

FundamentalsService

- `getFundamentals(ticker)` - Get fundamental data for single ticker

- `getBatchFundamentals(tickers)` - Get fundamentals for multiple tickers
- `enrichWithPrice(fundamentals, price)` - Calculate market cap, P/E, P/B ratios
- `enrichBatchWithPrices(fundamentals, priceMap)` - Batch enrich with prices
- `register(provider, config)` - Add fundamentals provider
- `checkProvidersHealth()` - Test all providers
- `getProviderStats()` - Get success/error statistics
- `clearCache()` - Clear fundamentals cache

SecEdgarProvider

- FREE - No API key required
- All US public companies
- Comprehensive US GAAP financial metrics
- Historical data back to ~2009
- Direct access to SEC filings (10-K, 10-Q)
- Smart caching (CIK: 30 days, Fundamentals: 90 days)
- Rate limiting (10 requests/second)
- ℹ Requires User-Agent header in format: "Company Name Email"

MarketstackProvider

- End-of-Day (EOD) stock prices
- 500,000+ tickers across 72+ exchanges worldwide
- Historical data with pagination
- Batch fetching support
- OHLCV data (Open, High, Low, Close, Volume)
- Company names included automatically
- ⚠ Requires API key (free tier: 100 requests/month)

CoinGeckoProvider

- Cryptocurrency prices (Bitcoin, Ethereum, 13M+ tokens)
- Current, historical, and intraday data
- 24/7 market data (crypto never closes)
- OHLCV data with market cap and volume
- Supports both ticker symbols (BTC, ETH) and CoinGecko IDs (bitcoin, ethereum)
- 240+ networks, 1600+ exchanges
- ℹ Optional API key (free tier: 30 requests/min, 10K/month)

OpenData

- `start()` - Initialize MongoDB connection
- `buildInitialDb()` - Import bulk data
- `CBusinessRecord` - Business record class
- `handelsregister` - German registry automation

Provider Architecture

Add custom data providers easily:

```
class MyCustomProvider implements IStockProvider {
  name = 'My Provider';
  priority = 50;
  requiresAuth = true;
  rateLimit = { requestsPerMinute: 60 };

  async fetchData(request: IStockDataRequest): Promise<IStockPrice | IStockPrice[]> {
    // Implement unified data fetching
    switch (request.type) {
      case 'current':
        return this.fetchCurrentPrice(request);
      case 'batch':
        return this.fetchBatchPrices(request);
      case 'historical':
        return this.fetchHistoricalPrices(request);
      default:
        throw new Error(`Unsupported request type`);
    }
  }

  async isAvailable(): Promise<boolean> {
    // Health check
    return true;
  }

  supportsMarket(market: string): boolean {
    return ['US', 'UK', 'DE'].includes(market);
  }

  supportsTicker(ticker: string): boolean {
    return /^[A-Z]{1,5}$/.test(ticker);
  }
}
```

```
stockService.register(new MyCustomProvider());
```

Performance

- **Batch Fetching**: Get 100+ prices in one API request
- **Smart Caching**: Data-type aware TTL (historical cached forever, EOD 24h, live 30s)
- **Rate Limit Management**: Automatic retry logic for API limits
- **Concurrent Processing**: Handle 1000+ records/second
- **Streaming**: Process GB-sized datasets without memory issues
- **Provider Fallback**: Automatic failover between data sources

Testing

Run the comprehensive test suite:

```
pnpm test
```

Test specific modules:

```
# Stock price providers
pnpm tstest test/test.marketstack.node.ts --verbose
pnpm tstest test/test.stocks.ts --verbose

# Fundamental data
pnpm tstest test/test.secedgar.provider.node.ts --verbose
pnpm tstest test/test.fundamentals.service.node.ts --verbose

# German business data
pnpm tstest test/test.handelsregister.ts --verbose
```

Getting API Keys

Marketstack (EOD Stock Data)

1. Visit marketstack.com

2. Sign up for a free account (100 requests/month)
3. Get your API key from the dashboard
4. Set environment variable: `MARKETSTACK_COM_TOKEN=your_key_here`

SEC EDGAR (Fundamental Data)

No API key required! SEC EDGAR is completely free and public. Just provide your company name and email in the User-Agent:

```
new SecEdgarProvider({
  userAgent: 'YourCompany youremail@example.com'
});
```

License and Legal Information

This repository contains open-source code that is licensed under the MIT License. A copy of the MIT License can be found in the license file within this repository.

Please note: The MIT License does not grant permission to use the trade names, trademarks, service marks, or product names of the project, except as required for reasonable and customary use in describing the origin of the work and reproducing the content of the NOTICE file.

Trademarks

This project is owned and maintained by Task Venture Capital GmbH. The names and logos associated with Task Venture Capital GmbH and any related products or services are trademarks of Task Venture Capital GmbH and are not included within the scope of the MIT license granted herein. Use of these trademarks must comply with Task Venture Capital GmbH's Trademark Guidelines, and any usage must be approved in writing by Task Venture Capital GmbH.

Company Information

Task Venture Capital GmbH Registered at District court Bremen HRB 35230 HB, Germany

For any legal inquiries or if you require further information, please contact us via email at hello@task.vc.

By using this repository, you acknowledge that you have read this section, agree to comply with its terms, and understand that the licensing of the code does not imply endorsement by Task Venture

Capital GmbH of any derivative works.

changelog.md for @fin.cx/opendata

2025-11-07 - 3.5.0 - feat(stocks)

Add provider fetch limits, intraday incremental fetch, cache deduplication, and provider safety/warning improvements

- Add maxRecords and defaultIntradayLimit to IProviderConfig to control maximum records per request and default intraday limits.
- CoinGecko provider: enforce maxRecords when processing historical data, warn when large historical/intraday results are returned without explicit limits, preserve priority mappings when rebuilding the coin cache, and improve cache load logging.
- Marketstack provider: make safety maxRecords configurable, apply a configurable default intraday limit, warn when no explicit limit is provided, and ensure effective limits are applied to returned results.
- StockPriceService: always attempt incremental fetch for intraday requests without a date to fetch only new data since the last cached timestamp and fall back to full fetch when necessary.
- StockPriceService: deduplicate price arrays by timestamp before caching and after merges to avoid duplicate timestamps and reduce cache bloat.
- Introduce StockDataService for unified access to prices and fundamentals with automatic enrichment (market cap, P/E, price-to-book) and caching improvements.
- Various cache/TTL improvements and safer default behaviors for intraday, historical and live data to improve performance and memory usage.

2025-11-06 - 3.4.0 - feat(stocks)

Introduce unified stock data service, new providers, improved caching and German business data tooling

- Add StockDataService: unified API to fetch price + fundamentals with automatic enrichment and batch support
- Introduce BaseProviderService abstraction and refactor provider management, caching and retry logic

- Enhance StockPriceService: unified getData, discriminated union request types, data-type aware TTLs and smarter cache keys
- Add Marketstack provider with intraday/EOD selection, pagination, OHLCV and exchange filtering
- Add CoinGecko provider with robust rate-limiting, coin ID resolution and crypto support (current, historical, intraday)
- Add SEC EDGAR fundamentals provider: CIK lookup, company facts parsing, rate limiting and caching
- Improve FundamentalsService: unified fetching, caching and enrichment helpers (enrichWithPrice, enrichBatchWithPrices)
- Enhance Yahoo provider and other provider mappings for better company metadata and market state handling
- Add German business data tooling: JsonlDataProcessor for JSONL bulk imports, HandelsRegister browser automation with download handling and parsing
- Expose OpenData entry points: DB init, JSONL processing and Handelsregister integration; add readme/docs and usage examples

2025-11-02 - 3.3.0 - feat(stocks/CoinGeckoProvider)

Add CoinGecko provider for cryptocurrency prices, export and tests, and update documentation

- Implemented CoinGeckoProvider with rate limiting, coin-id resolution, and support for current, batch, historical and intraday price endpoints
- Added unit/integration tests for CoinGecko: test/test.coingecko.node+bun+deno.ts
- Exported CoinGeckoProvider from ts/stocks/index.ts
- Updated README and readme.hints.md with CoinGecko usage, provider notes and examples
- Added .claude/settings.local.json with webfetch and bash permissions required for testing and CI

2025-11-01 - 3.2.2 - fix(handelsregister)

Correct screenshot path handling in HandelsRegister and add local tool permissions

- ts/classes.handelsregister.ts: Replace string concatenation for screenshot path with a template literal and explicit string assertion to ensure the path is formed correctly for page.screenshot() and avoid type issues.
- Add .claude/settings.local.json: Introduce local Claude settings that grant specific tool permissions used during development and testing (bash commands, web fetches, pnpm build, tstest, etc.).

2025-11-01 - 3.2.1 - fix(stocks/providers/provider.secedgar)

Improve SEC EDGAR provider networking and error handling, update plugin path import, bump dev deps and add/refresh tests and lockfile

- SEC EDGAR provider: switch from SmartRequest to native fetch for ticker list and company facts, add AbortController-based timeouts, handle gzip automatically, improve response validation and error messages, and keep CIK/ticker-list caching
- Improve timeout and rate-limit handling in SecEdgarProvider (uses native fetch + explicit timeout clear), plus clearer logging on failures
- Update ts/plugins import to use node:path for Node compatibility
- Bump devDependencies: @git.zone/tsrun to ^1.6.2 and @git.zone/tstest to ^2.7.0; bump @push.rocks/smartrequest to ^4.3.4
- Add and refresh comprehensive test files (node/bun/deno variants) for fundamentals, marketstack, secedgar and stockdata services
- Add deno.lock (dependency lock) and a local .claude/settings.local.json for CI/permissions

2025-11-01 - 3.2.0 - feat(StockDataService)

Add unified StockDataService and BaseProviderService with new stockdata interfaces, provider integrations, tests and README updates

- Introduce StockDataService: unified API to fetch prices and fundamentals with automatic enrichment and caching

- Add IStockData and IStockDataServiceConfig interfaces to define combined price+fundamentals payloads and configuration
- Implement BaseProviderService abstraction to share provider registration, health, stats and caching logic
- Add classes.stockdataservice.ts implementing batch/single fetch, enrichment, caching, health checks and provider stats
- Export new stockdata module and classes from ts/stocks/index.ts
- Add comprehensive tests: test/test.stockdata.service.node.ts to cover setup, provider registration, fetching, caching, enrichment, health and error handling
- Update README with Unified Stock Data API examples, usage, and documentation reflecting new unified service
- Minor infra: add .claude/settings.local.json permissions for local tooling and web fetch domains

2025-11-01 - 3.1.0 - feat(fundamentals)

Add FundamentalsService and SEC EDGAR provider with caching, rate-limiting, tests, and docs updates

- Introduce FundamentalsService to manage fundamentals providers, caching, retry logic and provider statistics
- Add SecEdgarProvider to fetch SEC EDGAR company facts (CIK lookup, company facts parsing) with rate limiting and local caches
- Expose fundamentals interfaces and services from ts/stocks (exports updated)
- Add comprehensive tests for FundamentalsService and SecEdgarProvider (new test files)
- Update README with new Fundamentals module documentation, usage examples, and configuration guidance
- Implement caching and TTL handling for fundamentals data and provider-specific cache TTL support
- Add .claude/settings.local.json (local permissions) and various test improvements

2025-10-31 - 3.0.0 - BREAKING CHANGE(stocks)

Unify stock provider API to discriminated IStockDataRequest and add company name/fullname enrichment

- Replace legacy provider methods (fetchPrice/fetchPrices) with a single fetchData(request: IStockDataRequest) on IStockProvider — providers must be migrated to the new signature.
- Migrate StockPriceService to the unified getData(request: IStockDataRequest) API. Convenience helpers getPrice/getPrices now wrap getData.
- Add companyName and companyFullName fields to IStockPrice and populate them in provider mappings (Marketstack mapping updated; Yahoo provider updated to support the unified API).
- MarketstackProvider: added buildCompanyFullName helper and improved mapping to include company identification fields and full name formatting.
- YahooFinanceProvider: updated to implement fetchData and to route current/batch requests through the new unified request types; historical/intraday throw explicit errors.
- Updated tests to exercise the new unified API, company-name enrichment, caching behavior, and provider direct methods.
- Note: This is a breaking change for external providers and integrations that implemented the old fetchPrice/fetchPrices API. Bump major version.

2025-10-31 - 2.1.0 - feat(stocks)

Add unified stock data API (getData) with historical/OHLCV support, smart caching and provider enhancements

- Introduce discriminated union request types (IStockDataRequest) and a unified getData() method (replaces legacy getPrice/getPrices for new use cases)
- Add OHLCV fields (open, high, low, volume, adjusted) and metadata (dataType, fetchedAt) to IStockPrice
- Implement data-type aware smart caching with TTLs (historical = never expire, EOD = 24h, live = 30s, intraday matches interval)
- Extend StockPriceService: new getData(), data-specific cache keys, cache maxEntries increased (default 10000), and TTL-aware add/get cache logic
- Enhance Marketstack provider: unified fetchData(), historical date-range retrieval with pagination, exchange filtering, batch current fetch, OHLCV mapping, and intraday placeholder
- Update Yahoo provider to include dataType and fetchedAt (live data) and maintain legacy fetchPrice/fetchPrices compatibility
- Add/adjust tests to cover unified API, historical retrieval, OHLCV presence and smart caching behavior; test setup updated to require explicit OpenData directory paths
- Update README to document v2.1 changes, migration examples, and new stock provider capabilities

2025-10-31 - 2.0.0 - BREAKING CHANGE(OpenData)

Require explicit directory paths for OpenData (nogit/download/germanBusinessData); remove automatic .nogit creation; update HandelsRegister, JsonlDataProcessor, tests and README.

- Breaking: OpenData constructor now requires a config object with nogitDir, downloadDir and germanBusinessDataDir. The constructor will throw if these paths are not provided.
- Removed automatic creation/export of .nogit/download/germanBusinessData from ts/paths. OpenData.start now ensures the required directories exist.
- HandelsRegister API changed: constructor now accepts downloadDir and manages its own unique download folder; screenshot and download paths now use the configured downloadDir.
- JsonlDataProcessor now accepts a germanBusinessDataDir parameter and uses it when ensuring/storing data instead of relying on global paths.
- Updated tests to provide explicit path configuration (tests now set testNogitDir, testDownloadDir, testGermanBusinessDataDir and write outputs accordingly) and to use updated constructors and qenv usage.
- Documentation updated (README) to document the breaking change and show examples for required directory configuration when instantiating OpenData.
- Added .claude/settings.local.json for local permissions/config used in development/CI environments.

2025-10-11 - 1.7.0 - feat(stocks)

Add Marketstack provider (EOD) with tests, exports and documentation updates

- Add MarketstackProvider implementation (ts/stocks/providers/provider.marketstack.ts) providing EOD single and batch fetching, availability checks and mapping to IStockPrice.
- Export MarketstackProvider from ts/stocks/index.ts so it is available via the public API.
- Add comprehensive Marketstack tests (test/test.marketstack.node.ts) covering registration, health checks, single/batch fetches, caching, ticker/market validation, provider stats and sample output.
- Update README with Marketstack usage examples, configuration, API key instructions and provider/health documentation.
- Bump dev dependency @git.zone/tstest to ^2.4.2 in package.json.
- Add project helper/config files (.claude/settings.local.json, .serena/project.yml and .serena/.gitignore) to support CI/tooling.

2025-09-24 - 1.6.1 - fix(stocks)

Fix Yahoo provider request handling and bump dependency versions

- Refactored Yahoo Finance provider to use SmartRequest.create() builder and await response.json() for HTTP responses (replaces direct getJson usage).
- Improved batch and single-price fetching to use the SmartRequest API, keeping User-Agent header and timeouts.
- Added a compile-time type-check alias to ensure IStockPrice matches tsclass.finance.IStockPrice.
- Bumped development and runtime dependency versions (notable bumps include @git.zone/tsbuild, @git.zone/tstest, @push.rocks/qenv, @push.rocks/smartarchive, @push.rocks/smartdata, @push.rocks/smartfile, @push.rocks/smartlog, @push.rocks/smartpath, @push.rocks/smartrequest, @tsclass/tsclass).
- Added .claude/settings.local.json to grant local CI permissions for a few Bash commands.

2025-07-12 - 1.6.0 - feat(readme)

Revamp documentation and package description for enhanced clarity

- Restructured README to highlight real-time stock data and German business data, streamlining quick start and advanced examples
- Updated package.json description to better reflect library capabilities
- Added .claude/settings.local.json to define permissions for external tools
- Refined code examples in tests and documentation for improved clarity and consistency

2025-04-09 - 1.5.3 - fix(test)

Await file writes in Handelsregister tests to ensure all downloads complete before test end

- Replaced array.map with await Promise.all to properly await asynchronous file writes in test/test.handelsregister.ts
- Improved robustness of asynchronous operations in test suite

2025-04-09 - 1.5.2 - fix(readme)

Improve .env configuration code block formatting in documentation

- Wrap the .env variables block in triple backticks for clarity
- Ensure consistency in the Markdown styling of code snippets

2025-04-09 - 1.5.1 - fix(core)

No changes detected in project files or documentation. This commit is a placeholder to record that nothing was updated.

2025-04-09 - 1.5.0 - feat(documentation)

Enhance project metadata and documentation with comprehensive usage examples, updated descriptions, and improved keywords.

- Updated npmextra.json and package.json to refine the project description and keyword list.
- Expanded readme.md with detailed sections on environment setup, CRUD operations, bulk JSONL processing, and advanced Handelsregister integrations.
- Included advanced workflow examples and error handling strategies in the documentation.
- Adjusted test cases (e.g. in test/test.handelsregister.ts) to reflect changes in company name usage.

2025-04-08 - 1.4.6 - fix(tests & jsonl)

Improve test structure and refine JSONL parsing for incomplete data

- Refactored test files to remove redundant get-specific-company tests in test.ts and added missing tests in test.handelsregister.ts
- Updated JSONL data processor to conditionally parse remaining data when available

2025-04-05 - 1.4.5 - fix(metadata)

Update repository, bugs, and homepage URLs to code.foss.global

- Repository URL updated from gitlab.com to code.foss.global
- Bugs URL updated from gitlab.com to code.foss.global
- Homepage URL updated to code.foss.global

2025-04-05 - 1.4.4 - fix(dependencies & tests)

Update dependency versions and refine test search query

- Bumped versions for several dependencies in package.json, including @git.zone/tsbuild, @git.zone/tsbundle, @git.zone/tstest, @push.rocks/tapbundle, @push.rocks/smartdata, @push.rocks/smartfile, @push.rocks/smartpromise, @push.rocks/smartrequest, and @tsclass/tsclass
- Updated test file to replace the search query 'Volkswagen' with 'LADR'
- Re-enabled the build initial data test by removing tap.skip

2025-01-07 - 1.4.3 - fix(test)

Corrected index value in test for fetching specific company data

- Updated the index from 8 to 7 for the germanParsedRegistration fetch in test

2025-01-07 - 1.4.2 - fix(core)

Fix concurrency and download handling in HandelsRegister class and adjust test cases

- Improved the clickFindButton function to include an argument for results limit.
- Enhanced the downloadFile function to rename and ensure files are correctly handled.
- Updated searchCompany method to allow specifying a limit on the number of search results.
- Adjusted test cases to select specific test data indices and output test files to a dedicated directory.

2025-01-04 - 1.4.1 - fix(core)

Fix issues with JSONL data processing and improve error handling in business record validation

- Fixed JSONL data processing by adding concurrent processing for each JSON line to enhance performance.
- Added validation logic in BusinessRecord class to ensure that the mandatory fields are checked.
- Adjusted environment variable loading in OpenData class to ensure correct database initialization.
- Included missing dependencies and exports in the project files to ensure proper functionality.

2025-01-04 - 1.4.0 - feat(HandelsRegister)

Add file download functionality to HandelsRegister

- Implemented file download feature in the HandelsRegister class.
- Configured pages in Puppeteer to allow downloads and set download paths.
- Parsed German registration information with more robust error handling.
- Added specific methods for downloading and handling 'SI' and 'AD' files.

2025-01-03 - 1.3.1 - fix(HandelsRegister)

Refined HandelsRegister functionality for better error handling and response capture.

- Improved parsing logic in parseGermanRegistration function.
- Enhanced navigateToPage and clickFindButton methods with error messages for clarity.
- Implemented a new responseListener to handle and log HTTP responses correctly.

2025-01-03 - 1.3.0 - feat(core)

Enhanced data handling capabilities and improved company search functionalities.

- Updated business record handling to support more registration types.
- Improved search capabilities for fetching company data with refined registration type matching.
- Added robust logging for JSONL data processing with early exit on successful parse.
- Reorganized test cases to include specific company data retrieval.

2025-01-02 - 1.2.1 - fix(BusinessRecord)

Add missing field registrationType to BusinessRecord data

- Introduced the 'registrationType' field to the BusinessRecord data schema with possible values 'HRA' or 'HRB'.

2025-01-02 - 1.2.0 - feat(core)

Integrate Handelsregister search for company data retrieval

- Added support for searching company data via Handelsregister.
- Replaced GermanBusinessData functionality with JsonlDataProcessor.
- Included smartbrowser dependency for handling web requests to Handelsregister.

2025-01-01 - 1.1.5 - fix(GermanBusinessData)

Add console log for total records processed at the end of the stream.

- Ensure that the number of records processed is logged at the end of data stream processing.

2024-12-31 - 1.1.4 - fix(documentation)

Update description and keywords in package.json

- Corrected the package description to reflect the focus on managing, accessing, and updating open data with MongoDB integration.
- Expanded the keywords in the package metadata to include data integration and processing terms.
- Improved README.md with more extensive setup, usage, and introduction of the library's functionalities.

2024-12-31 - 1.1.3 - fix(core)

Added missing license file for project completeness.

- Introduced a LICENSE file to the project, ensuring clarity on software usage permissions.

2024-12-31 - 1.1.2 - fix(GermanBusinessData)

Ensure unique ID generation for BusinessRecord

- Added generation of a new ID for each BusinessRecord in GermanBusinessData.
- Ensures each business record has a unique identifier.

2024-12-31 - 1.1.1 - fix(dependencies)

Update dependencies and devDependencies to newer versions.

- @git.zone/tsbuild from ^2.1.25 to ^2.2.0

- @git.zone/tsbundle from ^2.0.5 to ^2.1.0
- @git.zone/tsrun from ^1.2.46 to ^1.3.3
- @git.zone/tstest from ^1.0.84 to ^1.0.90
- @push.rocks/tapbundle from ^5.0.15 to ^5.5.4
- @types/node from ^20.9.0 to ^22.10.2
- @push.rocks/qenv from ^6.0.4 to ^6.1.0
- @push.rocks/smartarchive from ^4.0.19 to ^4.0.39
- @push.rocks/smartdata from ^5.0.33 to ^5.2.10
- @push.rocks/smartfile from ^11.0.0 to ^11.0.23
- @push.rocks/smartpath from ^5.0.11 to ^5.0.18
- @push.rocks/smartpromise from ^4.0.3 to ^4.0.4
- @push.rocks/smartrequest from ^2.0.21 to ^2.0.23
- @push.rocks/smartstream from ^3.0.30 to ^3.2.5

2024-12-31 - 1.1.0 - feat(core)

Enhanced data handling and retrieval features, improved usage documentation

- Updated package description and added project keywords in package.json.
- Extended readme with detailed usage examples and class structures.
- Added getBusinessRecordByName function for dynamic business record retrieval.

2024-07-02 - 1.0.3 - fix(core)

No new changes detected. Preparing for patch release.

2023-11-14 - 1.0.1 to 1.0.2 - General Updates

Minor version updates and fixes.

- fix(core): update