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.

Revision #4
Created 2026-03-28 10:49:18 UTC by foss.global Team
Updated 2026-03-28 12:14:43 UTC by foss.global Team