@fin.cx/skr

a module implementing german SKR03/SKR04 account structures.

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

readme.md for @fin.cx/skr

> **Enterprise-grade German accounting standards implementation for SKR03 and SKR04**
> Rock-solid double-entry bookkeeping with MongoDB persistence, e-invoice integration, and full TypeScript support

□□ Why @fin.cx/skr?

Building compliant German accounting software? You've come to the right place! This module provides a **complete, type-safe implementation** of the German standard charts of accounts (Standardkontenrahmen) SKR03 and SKR04, the backbone of professional accounting in Germany.

□□ What makes it awesome?

- □□ **Enterprise-Ready**: Production-tested implementation following HGB/GoBD standards
- ⚡ **Lightning Fast**: MongoDB-powered with optimized indexing and real-time balance updates
- □□ **Type-Safe**: Full TypeScript support with comprehensive type definitions
- □□ **Developer-Friendly**: Intuitive API that makes complex accounting operations simple
- □□ **Real-time Reporting**: Generate financial statements on-the-fly
- □□ **Transaction Safety**: Built-in double-entry validation and automatic reversals
- □ **Battle-Tested**: 65+ comprehensive tests covering all edge cases
- □□ **SKR Validation**: Automatic validation against official SKR standards
- □□ **E-Invoice Support**: Full XRechnung/ZUGFeRD integration for modern invoice processing
- □□ **Cryptographic Security**: Merkle tree and digital signature support for audit trails
- □□ **PDF Export**: Professional PDF report generation with customizable templates

□□ Installation

```
# Using npm
npm install @fin.cx/skr
```

```
# Using pnpm (recommended)
pnpm add @fin.cx/skr

# Using yarn
yarn add @fin.cx/skr
```

□□ Quick Start

Basic Setup

```
import { SkrApi } from '@fin.cx/skr';

// Initialize the API
const api = new SkrApi({
  mongoDbUrl: 'mongodb://localhost:27017',
  dbName: 'accounting' // optional, defaults to 'skr_accounting'
});

// Choose your SKR standard (SKR03 or SKR04)
await api.initialize('SKR03');
```

□□ Posting Transactions

```
// Simple transaction posting
const transaction = await api.postTransaction({
  date: new Date(),
  debitAccount: '1200',  // Bank account
  creditAccount: '8400', // Revenue account
  amount: 1190.00,
  description: 'Invoice #2024-001 payment received',
  reference: 'INV-2024-001',
  vatAmount: 190.00
});
```

```
// Complex journal entry with multiple lines
const journalEntry = await api.postJournalEntry({
  date: new Date(),
  description: 'Monthly salary payments',
  reference: 'SAL-2024-03',
  lines: [
    { accountNumber: '6000', debit: 5000.00, description: 'Gross salary' },
    { accountNumber: '6100', debit: 1000.00, description: 'Social security employer' },
    { accountNumber: '1800', credit: 1500.00, description: 'Tax withholding' },
    { accountNumber: '1200', credit: 4500.00, description: 'Net payment' }
  ]
});
```

□□ E-Invoice Integration

```
// Import electronic invoices (XRechnung/ZUGFeRD)
const invoiceData = await api.importInvoice(xmlContent, {
  format: 'xrechnung',
  validateSchema: true,
  checkDuplicates: true
});

// Automatically book invoice to accounting
const booking = await api.bookInvoice(invoiceData.invoiceId, {
  autoDetectAccounts: true,
  splitVAT: true,
  createPaymentSchedule: true
});

// Export invoice in various formats
const xRechnung = await api.exportInvoice(invoiceId, {
  format: 'xrechnung',
  version: '3.0',
  includeAttachments: true
});

// Search and filter invoices
const invoices = await api.searchInvoices({
```

```
  dateFrom: new Date('2024-01-01'),
  dateTo: new Date('2024-12-31'),
  status: 'booked',
  minAmount: 100,
  customerVATId: 'DE123456789'
});

// Generate compliance reports
const complianceReport = await api.createInvoiceComplianceReport({
  period: '2024-Q1',
  includeValidation: true,
  includeStatistics: true
});
```

□□ Generating Financial Reports

```
// Trial Balance (Summen- und Saldenliste)
const trialBalance = await api.generateTrialBalance({
  dateFrom: new Date('2024-01-01'),
  dateTo: new Date('2024-12-31')
});

// Income Statement (GuV - Gewinn- und Verlustrechnung)
const incomeStatement = await api.generateIncomeStatement({
  dateFrom: new Date('2024-01-01'),
  dateTo: new Date('2024-12-31')
});

// Balance Sheet (Bilanz)
const balanceSheet = await api.generateBalanceSheet({
  date: new Date('2024-12-31')
});

// General Ledger Export
const generalLedger = await api.generateGeneralLedger({
  dateFrom: new Date('2024-01-01'),
  dateTo: new Date('2024-12-31')
});
```

```
// Cash Flow Statement
const cashFlow = await api.generateCashFlowStatement({
  dateFrom: new Date('2024-01-01'),
  dateTo: new Date('2024-12-31')
});
```

□□ Advanced Export Features

```
// Export complete annual closing package (Jahresabschluss)
const jahresabschluss = await api.exportJahresabschluss({
  year: 2024,
  includeReports: ['balance_sheet', 'income_statement', 'cash_flow'],
  format: 'structured', // 'structured' | 'pdf' | 'csv'
  language: 'de',
  signatureRequired: true
});

// Generate PDF reports with professional formatting
const pdfReports = await api.generatePdfReports({
  reports: ['trial_balance', 'income_statement', 'balance_sheet'],
  dateFrom: new Date('2024-01-01'),
  dateTo: new Date('2024-12-31'),
  companyInfo: {
    name: 'Mustermann GmbH',
    address: 'Hauptstraße 1, 10115 Berlin',
    taxNumber: 'DE123456789',
    registrationNumber: 'HRB 12345'
  },
  outputPath: './reports/',
  template: 'professional' // Custom templates available
});

// Export with cryptographic signatures for audit trail
const signedExport = await api.signExport({
  data: jahresabschluss,
  privateKey: privateKeyPEM,
  certificate: certificatePEM,
```

```
  includeTimestamp: true,
  hashAlgorithm: 'SHA256'
});

// Detailed account data export
const accountExport = await api.exportAccountData({
  dateFrom: new Date('2024-01-01'),
  dateTo: new Date('2024-12-31'),
  format: 'detailed', // 'summary' | 'detailed' | 'tree'
  includeTransactions: true,
  includeBalances: true
});

// Balance history export for analysis
const balanceHistory = await api.exportBalanceData({
  accounts: ['1200', '1000', '8400'],
  interval: 'monthly', // 'daily' | 'weekly' | 'monthly' | 'quarterly'
  dateFrom: new Date('2024-01-01'),
  dateTo: new Date('2024-12-31'),
  includeRunningTotals: true
});

// Ledger export with filtering options
const ledgerExport = await api.exportLedgerData({
  accounts: ['1000-1999'], // Range support
  dateFrom: new Date('2024-01-01'),
  dateTo: new Date('2024-12-31'),
  includeReversals: false,
  groupByAccount: true,
  format: 'journal' // 'journal' | 'T-account' | 'chronological'
});
```

□□ Core Features

Account Management

```
// Create custom accounts
const account = await api.createAccount({
  accountNumber: '1299',
  accountName: 'PayPal Business',
  accountClass: 1,
  accountType: 'asset',
  description: 'PayPal business account for online payments',
  isActive: true
});

// Batch create multiple accounts for efficiency
const accounts = await api.createBatchAccounts([
  { accountNumber: '1298', accountName: 'Stripe Account', accountClass: 1, accountType:
'asset' },
  { accountNumber: '1297', accountName: 'Wise Business', accountClass: 1, accountType: 'asset'
}
]);

// Search accounts by name or number
const accounts = await api.searchAccounts('bank');

// Get account with full details
const account = await api.getAccount('1200');

// Update account information
await api.updateAccount('1200', {
  accountName: 'Main Business Bank Account',
  description: 'Primary operating account'
});

// Get account balance with running totals
const balance = await api.getAccountBalance('1200');
console.log(`Balance: €${balance.balance}`);
console.log(`Total Debits: €${balance.debitTotal}`);
console.log(`Total Credits: €${balance.creditTotal}`);

// List accounts by classification
const assetAccounts = await api.getAccountsByType('asset');
const class4Accounts = await api.getAccountsByClass(4);
```

```
// Paginated account access for large datasets
const pagedAccounts = await api.getAccountsPaginated({
  page: 1,
  limit: 50,
  sortBy: 'accountNumber',
  sortOrder: 'asc'
});
```

Transaction Management

```
// Get transaction by ID
const transaction = await api.getTransaction(transactionId);

// Get transaction history with filtering
const transactions = await api.listTransactions({
  accountNumber: '1200',
  dateFrom: new Date('2024-01-01'),
  dateTo: new Date('2024-12-31'),
  minAmount: 100,
  maxAmount: 10000
});

// Get all transactions for a specific account
const accountTransactions = await api.getAccountTransactions('1200', {
  dateFrom: new Date('2024-01-01'),
  dateTo: new Date('2024-12-31')
});

// Reverse transactions (Storno)
const reversal = await api.reverseTransaction(transactionId);

// Reverse complex journal entries
const journalReversal = await api.reverseJournalEntry(journalEntryId);

// Batch processing for performance
const batchResults = await api.postBatchTransactions([
  { date: new Date(), debitAccount: '1200', creditAccount: '8400', amount: 100 },
```

```
  { date: new Date(), debitAccount: '1200', creditAccount: '8400', amount: 200 },
  { date: new Date(), debitAccount: '1200', creditAccount: '8400', amount: 300 }
]);

// Paginated access for large datasets
const pagedTransactions = await api.getTransactionsPaginated({
  page: 1,
  limit: 50,
  sortBy: 'date',
  sortOrder: 'desc'
});

// Find unbalanced transactions for audit
const unbalanced = await api.getUnbalancedTransactions();
```

□□ SKR03 vs SKR04: Which One to Choose?

SKR03 - Process Structure Principle (Prozessgliederungsprinzip)

Best for: □□ Trading companies, □□ Service providers, □□ Retail businesses

- Accounts organized by **business process flow**
- Easier mapping to operational workflows
- Natural progression from purchasing → inventory → sales
- Popular with small to medium enterprises

SKR04 - Financial Classification Principle (Abschlussgliederungsprinzip)

Best for: □□ Manufacturing companies, □□ Large corporations, □□ Public companies

- Accounts organized by **financial statement structure**

- Direct mapping to balance sheet and P&L positions
- Simplified financial reporting and analysis
- Preferred by auditors and financial institutions

□□ Account Structure

Both SKR standards follow the same 4-digit hierarchical structure:

```
[0-9]    → Account Class (Kontenklasse)
 [0-9]   → Account Group (Kontengruppe)
  [0-9]  → Account Subgroup (Kontenuntergruppe)
   [0-9] → Individual Account (Einzelkonto)
```

Account Classes Overview

Class	SKR03 Description	SKR04 Description	Type
0	Fixed Assets (Anlagevermögen)	Fixed Assets	Asset
1	Current Assets (Umlaufvermögen)	Financial & Current Assets	Asset
2	Equity (Eigenkapital)	Expenses Part 1	Equity/Expense
3	Liabilities (Fremdkapital)	Expenses Part 2	Liability/Expense
4	Operating Income (Betriebliche Erträge)	Revenues Part 1	Revenue
5	Material Costs (Materialaufwand)	Revenues Part 2	Expense/Revenue
6	Operating Expenses (Betriebsaufwand)	Special Accounts	Expense
7	Other Costs (Weitere Aufwendungen)	Cost Accounting	Expense
8	Income (Erträge)	Free for Use (Custom)	Revenue
9	Closing Accounts (Abschlusskonten)	Equity & Closing	System

□□ Advanced Features

Period Management

```
// Close accounting period with automatic adjustments
await api.closePeriod('2024-01', {
  performYearEndAdjustments: true,
  generateReports: true
});

// Recalculate all account balances
await api.recalculateBalances();
```

Data Import/Export

```
// Import accounts from CSV
const importedCount = await api.importAccountsFromCSV(csvContent);

// Export accounts to CSV
const csvExport = await api.exportAccountsToCSV();

// Export to DATEV format (for tax advisors)
const datevExport = await api.exportToDATEV({
  dateFrom: new Date('2024-01-01'),
  dateTo: new Date('2024-12-31')
});

// Export reports to CSV
const reportCsv = await api.exportReportToCSV('income_statement', {
  dateFrom: new Date('2024-01-01'),
  dateTo: new Date('2024-12-31')
});
```

Validation & Integrity

```
// Find unbalanced transactions
const unbalanced = await api.getUnbalancedTransactions();
```

```
// Validate double-entry before posting
const isValid = await api.validateDoubleEntry({
  debitAccount: '1000',
  creditAccount: '8400',
  amount: 100
});

// The API automatically validates all journal entries
// Will throw error if entry is unbalanced
try {
  await api.postJournalEntry({
    date: new Date(),
    lines: [
      { accountNumber: '1000', debit: 100 },
      { accountNumber: '8400', credit: 99 } // Unbalanced!
    ]
  });
} catch (error) {
  console.error('Journal entry is not balanced!');
}
```

Invoice Processing & Compliance

```
// Get invoice statistics and analytics
const stats = await api.getInvoiceStatistics({
  dateFrom: new Date('2024-01-01'),
  dateTo: new Date('2024-12-31'),
  groupBy: 'month',
  includeVATAnalysis: true
});

// Generate invoices programmatically
const invoice = await api.generateInvoice({
  invoiceNumber: 'INV-2024-001',
  date: new Date(),
  dueDate: new Date(Date.now() + 30 * 24 * 60 * 60 * 1000),
  seller: {
    name: 'Your Company GmbH',
```

```
    vatId: 'DE123456789',
    address: 'Hauptstraße 1, 10115 Berlin'
  },
  buyer: {
    name: 'Customer AG',
    vatId: 'DE987654321',
    address: 'Kundenweg 5, 80331 München'
  },
  lines: [
    {
      description: 'Consulting Services',
      quantity: 10,
      unitPrice: 100,
      vatRate: 19
    }
  ]
});

// Validate invoice compliance
const validation = await api.validateInvoice(invoice, {
  standard: 'xrechnung',
  checkBusinessRules: true,
  checkVATRules: true
});
```

Utility Functions

```
// Get SKR type description for account classes
const classDesc = api.getAccountClassDescription(4);
// Returns: "Operating Income (SKR03)" or "Revenues Part 1 (SKR04)"

// Get current SKR type
const skrType = api.getSKRType(); // Returns: 'SKR03' or 'SKR04'
```

□□ Type Safety

Full TypeScript support with comprehensive type definitions:

```
import type {
  TSKRType,
  IAccountData,
  ITransactionData,
  IJournalEntry,
  IJournalEntryLine,
  ITrialBalanceReport,
  IIncomeStatement,
  IBalanceSheet,
  IAccountFilter,
  ITransactionFilter,
  IPaginationParams,
  IAccountBalance,
  ICashFlowStatement,
  IGeneralLedger,
  IInvoice,
  IInvoiceLine,
  IInvoiceParty,
  IBookingRules,
  IValidationResult
} from '@fin.cx/skr';

// All operations are fully typed
const account: IAccountData = {
  accountNumber: '1200',
  accountName: 'Bank Account',
  accountClass: 1,
  accountType: 'asset',
  skrType: 'SKR03',
  isActive: true
};

// TypeScript will catch errors at compile time
const filter: IAccountFilter = {
  accountType: 'asset',
  isActive: true,
  accountClass: 1
};
```

```
// Journal entries are validated at type level
const journalEntry: IJournalEntry = {
  date: new Date(),
  description: 'Year-end closing',
  lines: [
    { accountNumber: '8400', debit: 0, credit: 1000 },
    { accountNumber: '9000', debit: 1000, credit: 0 }
  ]
};
```

□□ Real-World Example: Complete Annual Closing

Here's how to perform a complete Jahresabschluss (annual financial closing):

```
import { SkrApi } from '@fin.cx/skr';

async function performJahresabschluss() {
  const api = new SkrApi({
    mongoDbUrl: process.env.MONGODB_URL!,
    dbName: 'company_accounting'
  });

  await api.initialize('SKR04'); // Using SKR04 for better reporting structure

  // 1. Post year-end adjustments
  const adjustments = await api.postJournalEntry({
    date: new Date('2024-12-31'),
    description: 'Jahresabschlussbuchungen',
    reference: 'JA-2024',
    lines: [
      // Depreciation (AfA)
      { accountNumber: '3700', debit: 10000, description: 'AfA auf Anlagen' },
      { accountNumber: '0210', credit: 10000, description: 'Wertberichtigung Gebäude' },

      // Provisions (Rückstellungen)
```

```
    { accountNumber: '3500', debit: 5000, description: 'Bildung Rückstellungen' },
    { accountNumber: '0800', credit: 5000, description: 'Sonstige Rückstellungen' },

    // VAT clearing
    { accountNumber: '1771', debit: 19000, description: 'USt-Saldo' },
    { accountNumber: '1571', credit: 17000, description: 'Vorsteuer-Saldo' },
    { accountNumber: '1700', credit: 2000, description: 'USt-Zahllast' }
  ]
});

// 2. Generate comprehensive annual closing package
const jahresabschluss = await api.exportJahresabschluss({
  year: 2024,
  includeReports: ['balance_sheet', 'income_statement', 'cash_flow', 'trial_balance'],
  format: 'pdf',
  language: 'de',
  signatureRequired: true,
  companyInfo: {
    name: 'Mustermann GmbH',
    address: 'Hauptstraße 1, 10115 Berlin',
    taxNumber: 'DE123456789',
    registrationNumber: 'HRB 12345'
  }
});

// 3. Generate individual reports for analysis
const incomeStatement = await api.generateIncomeStatement({
  dateFrom: new Date('2024-01-01'),
  dateTo: new Date('2024-12-31')
});

const balanceSheet = await api.generateBalanceSheet({
  date: new Date('2024-12-31')
});

const cashFlow = await api.generateCashFlowStatement({
  dateFrom: new Date('2024-01-01'),
  dateTo: new Date('2024-12-31')
});
```

```
  // 4. Export for tax advisor in DATEV format
  const datevExport = await api.exportToDATEV({
    dateFrom: new Date('2024-01-01'),
    dateTo: new Date('2024-12-31')
  });

  // 5. Create signed export for audit trail
  const signedExport = await api.signExport({
    data: jahresabschluss,
    privateKey: process.env.PRIVATE_KEY!,
    certificate: process.env.CERTIFICATE!,
    includeTimestamp: true
  });

  // 6. Close the period
  await api.closePeriod('2024-12', {
    performYearEndAdjustments: true,
    generateReports: true
  });

  console.log('□□Jahresabschluss 2024 Complete!');
  console.log(`□□Umsatz: €${incomeStatement.totalRevenue.toLocaleString('de-DE')}`);
  console.log(`□□Aufwendungen: €${incomeStatement.totalExpenses.toLocaleString('de-DE')}`);
  console.log(`□□Jahresergebnis: €${incomeStatement.netIncome.toLocaleString('de-DE')}`);
  console.log(`□□Bilanzsumme: €${balanceSheet.assets.totalAssets.toLocaleString('de-DE')}`);
  console.log(`□□Cash Flow: €${cashFlow.netCashFlow.toLocaleString('de-DE')}`);
  console.log(incomeStatement.netIncome > 0 ? '□ Gewinn!' : '□□Verlust');

  await api.close();
}

performJahresabschluss().catch(console.error);
```

□□ API Reference

Main Classes

Class	Description
`SkrApi`	Main API entry point for all operations
`ChartOfAccounts`	Account management and initialization
`Ledger`	General ledger and transaction posting with SKR validation
`Reports`	Financial reporting and exports
`Account`	Account model with balance tracking
`Transaction`	Double-entry transaction model
`JournalEntry`	Complex multi-line journal entries
`InvoiceAdapter`	XRechnung/ZUGFeRD invoice processing
`InvoiceBookingEngine`	Automatic invoice to accounting booking
`InvoiceStorage`	Invoice persistence and search

Key Methods

Method	Description
`initialize(skrType)`	Initialize with SKR03 or SKR04
`postTransaction(data)`	Post a simple two-line transaction
`postJournalEntry(data)`	Post complex multi-line journal entry
`postBatchTransactions(transactions)`	Post multiple transactions efficiently
`reverseTransaction(id)`	Create reversal (Storno) entry
`reverseJournalEntry(id)`	Reverse complex journal entries
`generateTrialBalance(params)`	Generate Summen- und Saldenliste
`generateIncomeStatement(params)`	Generate GuV (P&L) statement
`generateBalanceSheet(params)`	Generate Bilanz (balance sheet)
`generateCashFlowStatement(params)`	Generate cash flow statement
`generateGeneralLedger(params)`	Generate complete general ledger
`exportToDATEV(params)`	Export DATEV-compatible data
`exportJahresabschluss(params)`	Export complete annual closing package
`generatePdfReports(params)`	Generate professional PDF reports
`signExport(data)`	Create cryptographically signed exports
`importInvoice(data, options)`	Import XRechnung/ZUGFeRD invoices
`bookInvoice(invoiceId, rules)`	Book invoice to accounting

Method	Description
`exportInvoice(id, options)`	Export invoice in various formats
`searchInvoices(filter)`	Search and filter invoices
`closePeriod(period, options)`	Close accounting period
`recalculateBalances()`	Recalculate all account balances
`validateDoubleEntry(data)`	Validate transaction before posting
`getUnbalancedTransactions()`	Find integrity issues
`createBatchAccounts(accounts)`	Create multiple accounts at once

□□ Why Developers Love It

- □□ **Zero Configuration**: Pre-configured SKR03/SKR04 accounts out of the box
- □□ **Automatic Validation**: Never worry about unbalanced entries or wrong account types
- □□ **Real-time Analytics**: Instant financial insights with live balance updates
- □□ **SKR Compliance**: Validates against official SKR standards automatically
- □□ **High Performance**: Optimized MongoDB queries and batch operations
- □□ **German Compliance**: Full HGB/GoBD compliance built-in
- □□ **Type Safety**: Complete TypeScript definitions prevent runtime errors
- □□ **Smart Validation**: Warns about non-standard accounts and type mismatches
- □□ **E-Invoice Ready**: Native XRechnung/ZUGFeRD support for modern workflows
- □□ **Audit-Proof**: Cryptographic signatures and Merkle trees for tamper-proof records
- □□ **Professional Reports**: Generate PDF reports that impress auditors and stakeholders

□□ Requirements

- **Node.js** >= 18.0.0
- **MongoDB** >= 5.0
- **TypeScript** >= 5.0 (for development)

□□ Testing

The module includes comprehensive test coverage with real-world scenarios:

```
# Run all tests
pnpm test

# Run specific test suites
pnpm test test/test.skr03.ts         # SKR03 functionality
pnpm test test/test.skr04.ts         # SKR04 functionality
pnpm test test/test.jahresabschluss.skr03.ts  # Annual closing SKR03
pnpm test test/test.jahresabschluss.skr04.ts  # Annual closing SKR04
pnpm test test/test.invoice.ts       # Invoice processing
pnpm test test/test.export.ts        # Export functionality
```

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/skr

2025-10-28 - 1.2.1 - fix(skr.classes.account)

Remove incorrect SKR04 automatic account 3300; improve VAT posting validation and test isolation; update readme hints and CI settings

- ts/skr.classes.account.ts: Removed account '3300' from the SKR04 automatic accounts list (3300 is Fahrzeugkosten and must be postable).
- ts/skr.postingkeys.ts: Relax VAT amount requirement — VAT amount is no longer required when posting to VAT accounts or to debtor/creditor accounts (settlement lines).
- ts/skr.classes.journalentry.ts: Detect VAT lines in journal entries and pass VAT-aware context into posting key validation to avoid false-positive VAT errors.
- test/test.skr04.ts: Use timestamped database names to ensure isolated test runs and avoid DB conflicts during CI.
- readme.hints.md: Updated status and notes (tests passing, recent fixes, architecture notes and validation pipeline).
- .claude/settings.local.json: Added local CI/agent permission settings used by the project environment.

All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog, and this project adheres to Semantic Versioning.

[1.2.0] - 2025-01-09

Added

- **E-Invoice Integration**: Full XRechnung/ZUGFeRD support with import/export capabilities
- **Invoice Processing**: Automatic booking of electronic invoices to accounting

- **Advanced Export Features**: Comprehensive export functionality for accounts, balances, and ledger data
- **PDF Generation**: Professional PDF report generation with customizable templates
- **Security Features**: Merkle tree audit trails and digital signature support for tamper-proof records
- **Invoice Storage**: Dedicated invoice persistence layer with search and filtering
- **Invoice Adapter**: Bidirectional conversion between e-invoice formats and internal data model
- **Invoice Booking Engine**: Intelligent automatic account detection and VAT splitting
- **Cryptographic Signatures**: Support for signing exports with private keys and certificates
- **Structured Export Formats**: Export data in multiple formats (JSON, CSV, PDF)
- **Jahresabschluss Export**: Complete annual closing package generation
- New dependencies: @e-invoice-eu/core, @fin.cx/einvoice, merkletreejs, node-forge
- Enhanced documentation with invoice and export examples

Changed

- Updated README with comprehensive documentation of new features
- Expanded API reference with new invoice and export methods

[1.1.0] - 2025-01-09

Added

- SKR standard validation in postJournalEntry to ensure accounts match official SKR03/SKR04 data
- Module-level Maps for O(1) SKR standard lookups
- validateAccountsAgainstSKR method for checking account type and class compliance
- Smart validation that allows SKR04 class 8 custom accounts
- Warning logs for non-standard accounts and type/class mismatches

Fixed

- Test isolation issues by adding timestamps to database names
- SKR04 test using correct account mappings (9xxx equity accounts)

Changed

- Enhanced README with accurate API documentation and testing instructions
- Updated legal section to Task Venture Capital GmbH

[1.0.0] - 2025-01-09

Added

- Initial release of @fin.cx/skr module
- Complete SKR03 implementation (Process Structure Principle)
- Complete SKR04 implementation (Financial Classification Principle)
- Double-entry bookkeeping validation system
- MongoDB persistence layer using @push.rocks/smartdata
- Comprehensive account management (100+ predefined accounts per SKR standard)
- Transaction posting and reversal capabilities
- Journal entry support with multiple lines
- Financial reporting suite:
  - Trial Balance generation
  - Income Statement (P&L)
  - Balance Sheet
  - General Ledger
  - Cash Flow Statement
- DATEV-compatible export formats
- Full TypeScript support with comprehensive type definitions
- API layer for external integration
- CSV import/export functionality
- VAT handling and cost center tracking
- Automatic balance calculations
- Period closing functionality
- Batch transaction processing

Technical Features

- Type-safe database operations
- Indexed MongoDB collections for performance
- Transaction atomicity and consistency
- Comprehensive validation rules
- 4-digit account number validation
- Account class hierarchy (0-9)
- Support for custom accounts
- Real-time balance updates