Daily Report Form - Developer Documentation

📋 Project Overview

This is a **Daily Sales & Financial Report Form** for Phil's Philly Steaks restaurant chain. The system captures daily transactions, sales data, and performs automatic financial calculations for end-of-day reconciliation.

🏗️ File Structure & Architecture

1. Form Layout Structure

```
Daily Report Form
├── Header Section
│   ├── Title: "Daily Report"
│   ├── Company Name: "Phil's Philly Steaks - Hulen Mall"
│   └── Company Info (Address, Phone)
├── Transaction Expenses Section
│   ├── Transaction Table (ID, Company, Type, Amount)
│   ├── Side Panel (Date, Weather, Events)
│   └── Category Labels (Accounting, Food Cost, Rent, Taxes)
├── Sales Section
│   ├── Projected Sales Input
│   ├── Cancels & Voids Tracking
│   └── No Sales Counter
└── Financial Summary Section
    ├── Left Panel (Sales Calculations)
    └── Right Panel (Cash Reconciliation)
```

💰 Business Logic & Calculations

Core Formulas Identified:

1. Transaction Totals

```javascript
// Total Paid Outs = Sum of all transaction amounts
totalPaidOuts = sum(transactionAmounts)
// Example: Icee Company (204) = 204 total
```

2. Sales Calculations

```javascript
// Net Sales = Gross Sales - Coupons - Adjustments
netSales = grossSales - couponsReceived - adjustments
// Example: 1126.45 - 3.92 - 4.32 = 1118.21

// Tax Calculation (Texas rate: 8.25%)
tax = netSales - (netSales / 1.0825)
// Example: 1118.21 - (1118.21 / 1.0825) = 85.22

// Sales (Pre-tax) = Net Sales - Tax
salesPreTax = netSales - tax
// Example: 1118.21 - 85.22 = 1032.99
```

3. Cash Reconciliation

```javascript
// Cash To Account For = Net Sales - Paid Outs - Credit Cards
cashToAccountFor = netSales - totalPaidOuts - creditCards
// Example: 1118.21 - 204 - 504.91 = 409.30

// Short/Over Calculation
if (actualDeposit < cashToAccountFor) {
    short = actualDeposit - cashToAccountFor  // Negative value
    over = 0
} else {
    short = 0
    over = actualDeposit - cashToAccountFor   // Positive value
}
// Example: 409 < 409.30, so Short = -0.30, Over = 0
```

🗃️ Data Model

Transaction Object

```typescript
```

```typescript
interface Transaction {
    id: number;
    company: string;
    transactionType: 'Food Cost' | 'Rent' | 'Accounting' | 'Taxes' | 'Other';
    amount: number;
}

// Example:
const transaction = {
    id: 1,
    company: "Icee Company",
    transactionType: "Food Cost",
    amount: 204.00
}
```

Daily Report Object

typescript

```typescript
interface DailyReport {
  // Header Info
  restaurantName: string;
  address: string;
  phone: string;
  reportDate: Date;
  pageNumber: number;

  // Environmental Data
  weather: string;
  holidayEvent: string;

  // Transactions
  transactions: Transaction[];

  // Sales Data
  projectedSales: number;
  grossSales: number;
  amountOfCancels: number;
  amountOfVoids: number;
  numberOfNoSales: number;

  // Coupons & Adjustments
  totalCoupons: number;
  couponsReceived: number;
  adjustmentsOverrings: number;

  // Customer Data
  totalCustomers: number;

  // Payment Data
  creditCards: number;
  actualDeposit: number;

  // Calculated Fields (Auto-computed)
  totalPaidOuts: number;     // Calculated
  netSales: number;          // Calculated
  tax: number;               // Calculated
  salesPreTax: number;       // Calculated
  cashToAccountFor: number;  // Calculated
  short: number;             // Calculated
  over: number;              // Calculated
```

```
    averageTicket: number;      // Calculated
}
```

🔧 Implementation Guide

1. Database Schema

```sql
-- Daily Reports Table
CREATE TABLE daily_reports (
    id SERIAL PRIMARY KEY,
    restaurant_name VARCHAR(100),
    report_date DATE NOT NULL,
    weather VARCHAR(50),
    holiday_event VARCHAR(100),
    projected_sales DECIMAL(10,2),
    gross_sales DECIMAL(10,2),
    amount_of_cancels DECIMAL(10,2),
    amount_of_voids DECIMAL(10,2),
    number_of_no_sales INTEGER,
    total_coupons INTEGER,
    coupons_received DECIMAL(10,2),
    adjustments_overrings DECIMAL(10,2),
    total_customers INTEGER,
    credit_cards DECIMAL(10,2),
    actual_deposit DECIMAL(10,2),
    created_at TIMESTAMP DEFAULT NOW()
);

-- Transactions Table
CREATE TABLE transactions (
    id SERIAL PRIMARY KEY,
    daily_report_id INTEGER REFERENCES daily_reports(id),
    transaction_id INTEGER,
    company VARCHAR(100),
    transaction_type VARCHAR(50),
    amount DECIMAL(10,2),
    created_at TIMESTAMP DEFAULT NOW()
);
```

2. API Endpoints

POST /api/daily-reports

javascript

```
// Create new daily report
{
  "restaurantName": "Phil's Philly Steaks - Hulen Mall",
  "reportDate": "2025-08-28",
  "weather": "Sunny",
  "grossSales": 1126.45,
  "transactions": [
    {
      "transactionId": 1,
      "company": "Icee Company",
      "transactionType": "Food Cost",
      "amount": 204.00
    }
  ]
}
```

GET /api/daily-reports/:date

javascript

```
// Retrieve daily report with calculated fields
{
  "id": 1,
  "reportDate": "2025-08-28",
  "grossSales": 1126.45,
  "netSales": 1118.21,       // Calculated
  "totalPaidOuts": 204.00,   // Calculated
  "cashToAccountFor": 409.30, // Calculated
  "short": -0.30,            // Calculated
  "over": 0.00               // Calculated
}
```

3. Frontend Implementation (React)

Component Structure

```
DailyReportForm/
├── components/
│   ├── HeaderSection.tsx
│   ├── TransactionTable.tsx
│   ├── SidePanel.tsx
│   ├── SalesSection.tsx
│   ├── FinancialSummary.tsx
│   └── CalculatedFields.tsx
├── hooks/
│   ├── useCalculations.ts
│   ├── useDailyReport.ts
│   └── useTransactions.ts
├── services/
│   └── dailyReportAPI.ts
└── types/
    └── dailyReport.ts
```

Key React Hook - useCalculations.ts

typescript

```
export const useCalculations = (reportData: DailyReportData) => {
  const calculations = useMemo(() => {
    const totalPaidOuts = reportData.transactions
      .reduce((sum, t) => sum + t.amount, 0);

    const netSales = reportData.grossSales -
                     reportData.couponsReceived -
                     reportData.adjustmentsOverrings;

    const tax = netSales - (netSales / 1.0825);

    const cashToAccountFor = netSales - totalPaidOuts - reportData.creditCards;

    const short = reportData.actualDeposit < cashToAccountFor ?
                  reportData.actualDeposit - cashToAccountFor : 0;

    const over = reportData.actualDeposit > cashToAccountFor ?
                 reportData.actualDeposit - cashToAccountFor : 0;

    const averageTicket = reportData.totalCustomers > 0 ?
                          netSales / reportData.totalCustomers : 0;

    return {
      totalPaidOuts,
      netSales,
      tax,
      salesPreTax: netSales - tax,
      cashToAccountFor,
      short,
      over,
      averageTicket
    };
  }, [reportData]);

  return calculations;
};
```

4. Validation Rules

typescript

```
const validationSchema = {
    grossSales: {
        required: true,
        min: 0,
        type: 'number'
    },
    projectedSales: {
        required: true,
        min: 0,
        type: 'number'
    },
    transactions: {
        required: true,
        minLength: 1,
        validate: (transactions) =>
            transactions.every(t => t.amount >= 0)
    },
    actualDeposit: {
        required: true,
        min: 0,
        type: 'number'
    }
};
```

🎯 Key Features to Implement

1. Auto-Calculations

- Real-time calculation updates as user inputs data
- Formula validation and error handling
- Rounding to 2 decimal places for currency

2. Data Validation

- Required field validation
- Numeric field validation
- Business logic validation (e.g., voids can't exceed sales)

3. Data Persistence

- Auto-save functionality
- Draft saving capability

- History/audit trail

4. Reporting Features

- Print-friendly format
- PDF export
- Daily/weekly/monthly summaries

5. Error Handling

- Calculation error detection
- Data inconsistency warnings
- User-friendly error messages

🔍 Business Rules & Constraints

1. **Transaction Types**: Limited to predefined categories
2. **Tax Rate**: Fixed at 8.25% (Texas rate)
3. **Currency**: All amounts in USD, rounded to 2 decimal places
4. **Date**: One report per restaurant per day
5. **Negative Values**: Only allowed for voids and adjustments
6. **Cash Reconciliation**: Must balance within $1.00 tolerance

🚀 Deployment Considerations

Environment Variables

```env
TAX_RATE=0.0825
CURRENCY_LOCALE=en-US
MAX_TRANSACTIONS_PER_DAY=100
CASH_TOLERANCE=1.00
```

Performance Optimization

- Index on `report_date` and `restaurant_name`
- Cache calculated fields
- Implement pagination for transaction history
- Use debouncing for real-time calculations

📊 Sample Test Data

```javascript
const sampleDailyReport = {
    restaurantName: "Phil's Philly Steaks - Hulen Mall",
    reportDate: "2025-08-28",
    weather: "Sunny",
    projectedSales: 1200.00,
    grossSales: 1126.45,
    amountOfCancels: 12.53,
    amountOfVoids: -136.23,
    numberOfNoSales: 7,
    couponsReceived: 3.92,
    adjustmentsOverrings: 4.32,
    totalCustomers: 45,
    creditCards: 504.91,
    actualDeposit: 409.00,
    transactions: [
        {
            transactionId: 1,
            company: "Icee Company",
            transactionType: "Food Cost",
            amount: 204.00
        }
    ]
};
```

This documentation provides a complete foundation for developers to understand the business logic, implement the calculations, and build a robust daily reporting system.