Mingus Key Date Feature

**Mingus Special Events & Milestone System**

*Database Setup, Variables, and Workflow for Milestone Financial Planning*

**Database Schema Design**

**New Table: user\_special\_events**

CREATE TABLE user\_special\_events (

id uuid PRIMARY KEY DEFAULT gen\_random\_uuid(),

user\_id uuid REFERENCES auth.users(id) NOT NULL,

event\_name text NOT NULL,

event\_type text NOT NULL CHECK (event\_type IN (

'child\_birthday',

'spouse\_birthday',

'parent\_birthday',

'anniversary',

'solo\_vacation',

'group\_trip\_girls',

'group\_trip\_boys',

'tax\_refund',

'property\_tax',

'car\_registration',

'other\_milestone'

)),

base\_date date NOT NULL, -- The original/first occurrence date

estimated\_amount decimal(10,2) NOT NULL,

frequency text DEFAULT 'annually' CHECK (frequency IN ('annually', 'one\_time')),

is\_active boolean DEFAULT true,

priority\_level integer DEFAULT 3 CHECK (priority\_level BETWEEN 1 AND 5), -- 1=critical, 5=nice-to-have

notes text,

created\_at timestamp DEFAULT now(),

updated\_at timestamp DEFAULT now()

);

-- Indexes for performance

CREATE INDEX idx\_user\_special\_events\_user\_id ON user\_special\_events(user\_id);

CREATE INDEX idx\_user\_special\_events\_type ON user\_special\_events(user\_id, event\_type);

CREATE INDEX idx\_user\_special\_events\_active ON user\_special\_events(user\_id, is\_active);

**New Table: user\_special\_event\_dates**

CREATE TABLE user\_special\_event\_dates (

id uuid PRIMARY KEY DEFAULT gen\_random\_uuid(),

user\_id uuid REFERENCES auth.users(id) NOT NULL,

special\_event\_id uuid REFERENCES user\_special\_events(id) ON DELETE CASCADE,

due\_date date NOT NULL,

amount decimal(10,2) NOT NULL,

event\_type text NOT NULL,

event\_name text NOT NULL,

priority\_level integer NOT NULL,

is\_covered boolean DEFAULT false, -- Will be calculated based on cash flow

coverage\_status text CHECK (coverage\_status IN ('covered', 'at\_risk', 'uncovered')),

created\_at timestamp DEFAULT now(),

updated\_at timestamp DEFAULT now(),

UNIQUE(user\_id, special\_event\_id, due\_date)

);

-- Indexes for performance

CREATE INDEX idx\_special\_event\_dates\_user\_date ON user\_special\_event\_dates(user\_id, due\_date);

CREATE INDEX idx\_special\_event\_dates\_coverage ON user\_special\_event\_dates(user\_id, coverage\_status);

**Enhanced daily\_cashflow Table**

-- Add new columns to existing daily\_cashflow table

ALTER TABLE daily\_cashflow

ADD COLUMN special\_events\_expense decimal(10,2) DEFAULT 0.00,

ADD COLUMN uncovered\_events\_count integer DEFAULT 0,

ADD COLUMN at\_risk\_events\_count integer DEFAULT 0;

-- Updated structure

CREATE TABLE daily\_cashflow\_enhanced (

id uuid PRIMARY KEY DEFAULT gen\_random\_uuid(),

user\_id uuid REFERENCES auth.users(id) NOT NULL,

forecast\_date date NOT NULL,

opening\_balance decimal(12,2) NOT NULL,

total\_income decimal(12,2) DEFAULT 0.00,

total\_expenses decimal(12,2) DEFAULT 0.00,

special\_events\_expense decimal(12,2) DEFAULT 0.00, -- NEW

closing\_balance decimal(12,2) NOT NULL,

net\_daily\_change decimal(12,2) NOT NULL,

balance\_status text CHECK (balance\_status IN ('positive', 'negative', 'critical')),

uncovered\_events\_count integer DEFAULT 0, -- NEW

at\_risk\_events\_count integer DEFAULT 0, -- NEW

created\_at timestamp DEFAULT now(),

updated\_at timestamp DEFAULT now(),

UNIQUE(user\_id, forecast\_date)

);

**Enhanced Python Backend System**

**Special Events Schedule Generator**

from datetime import datetime, timedelta

from dateutil.rrule import rrule, YEARLY

from dateutil.relativedelta import relativedelta

import logging

class SpecialEventsEngine:

def \_\_init\_\_(self, supabase\_client):

self.supabase = supabase\_client

def generate\_special\_event\_schedule(self, event\_data, forecast\_years=3):

"""

Generate recurring dates for special events

Args:

event\_data: Dict with base\_date, event\_type, amount, etc.

forecast\_years: How many years ahead to generate dates

Returns:

List of event occurrences

"""

base\_date = datetime.strptime(event\_data['base\_date'], "%Y-%m-%d")

end\_date = base\_date + timedelta(days=365 \* forecast\_years)

if event\_data['frequency'] == 'one\_time':

# One-time events only occur once

if base\_date >= datetime.now():

return [{

'due\_date': base\_date.strftime("%Y-%m-%d"),

'amount': event\_data['estimated\_amount'],

'event\_type': event\_data['event\_type'],

'event\_name': event\_data['event\_name'],

'priority\_level': event\_data['priority\_level']

}]

else:

return []

# Annual recurring events

annual\_dates = rrule(YEARLY, dtstart=base\_date, until=end\_date)

schedule = []

for event\_date in annual\_dates:

# Skip past dates

if event\_date >= datetime.now():

schedule.append({

'due\_date': event\_date.strftime("%Y-%m-%d"),

'amount': event\_data['estimated\_amount'],

'event\_type': event\_data['event\_type'],

'event\_name': event\_data['event\_name'],

'priority\_level': event\_data['priority\_level']

})

return schedule

async def populate\_special\_event\_dates(self, user\_id):

"""Populate user\_special\_event\_dates table for all user's special events"""

try:

# Get all active special events for user

events\_response = self.supabase.table('user\_special\_events')\

.select('\*')\

.eq('user\_id', user\_id)\

.eq('is\_active', True)\

.execute()

# Clear existing future dates

future\_date = datetime.now().strftime("%Y-%m-%d")

self.supabase.table('user\_special\_event\_dates')\

.delete()\

.eq('user\_id', user\_id)\

.gte('due\_date', future\_date)\

.execute()

# Generate dates for each event

for event in events\_response.data:

schedule = self.generate\_special\_event\_schedule(event)

for occurrence in schedule:

self.supabase.table('user\_special\_event\_dates').insert({

'user\_id': user\_id,

'special\_event\_id': event['id'],

'due\_date': occurrence['due\_date'],

'amount': occurrence['amount'],

'event\_type': occurrence['event\_type'],

'event\_name': occurrence['event\_name'],

'priority\_level': occurrence['priority\_level'],

'coverage\_status': 'uncovered' # Will be updated during cash flow calculation

}).execute()

return True

except Exception as e:

logging.error(f"Error populating special event dates for user {user\_id}: {e}")

return False

async def calculate\_event\_coverage(self, user\_id, days\_ahead=365):

"""

Calculate which special events are covered by available cash flow

Updates coverage\_status for each event

"""

try:

# Get daily cash flow data

start\_date = datetime.now().strftime("%Y-%m-%d")

end\_date = (datetime.now() + timedelta(days=days\_ahead)).strftime("%Y-%m-%d")

cashflow\_response = self.supabase.table('daily\_cashflow')\

.select('\*')\

.eq('user\_id', user\_id)\

.gte('forecast\_date', start\_date)\

.lte('forecast\_date', end\_date)\

.order('forecast\_date')\

.execute()

# Get special events in same period

events\_response = self.supabase.table('user\_special\_event\_dates')\

.select('\*')\

.eq('user\_id', user\_id)\

.gte('due\_date', start\_date)\

.lte('due\_date', end\_date)\

.order('due\_date')\

.execute()

# Create cashflow lookup

cashflow\_lookup = {row['forecast\_date']: row for row in cashflow\_response.data}

# Evaluate each event

coverage\_updates = []

for event in events\_response.data:

event\_date = event['due\_date']

event\_amount = event['amount']

if event\_date in cashflow\_lookup:

available\_balance = cashflow\_lookup[event\_date]['closing\_balance']

# Determine coverage status

if available\_balance >= event\_amount:

coverage\_status = 'covered'

is\_covered = True

elif available\_balance >= (event\_amount \* 0.8): # 80% threshold

coverage\_status = 'at\_risk'

is\_covered = False

else:

coverage\_status = 'uncovered'

is\_covered = False

# Update the event

self.supabase.table('user\_special\_event\_dates')\

.update({

'coverage\_status': coverage\_status,

'is\_covered': is\_covered

})\

.eq('id', event['id'])\

.execute()

coverage\_updates.append({

'event\_id': event['id'],

'event\_name': event['event\_name'],

'due\_date': event\_date,

'amount': event\_amount,

'available\_balance': available\_balance,

'coverage\_status': coverage\_status

})

return coverage\_updates

except Exception as e:

logging.error(f"Error calculating event coverage for user {user\_id}: {e}")

return []

# Enhanced Cash Flow Calculator

class EnhancedCashFlowEngine:

def \_\_init\_\_(self, supabase\_client):

self.supabase = supabase\_client

self.special\_events = SpecialEventsEngine(supabase\_client)

async def calculate\_daily\_cashflow\_with\_events(self, user\_id, days\_ahead=365):

"""Enhanced daily cash flow calculation including special events"""

try:

# Get opening balance

user\_response = self.supabase.table('users')\

.select('opening\_balance')\

.eq('id', user\_id)\

.execute()

opening\_balance = user\_response.data[0].get('opening\_balance', 0)

# Get date range

start\_date = datetime.now()

end\_date = start\_date + timedelta(days=days\_ahead)

# Get regular income/expenses (existing logic)

# ... [previous income/expense logic] ...

# Get special events

special\_events\_response = self.supabase.table('user\_special\_event\_dates')\

.select('\*')\

.eq('user\_id', user\_id)\

.gte('due\_date', start\_date.strftime("%Y-%m-%d"))\

.lte('due\_date', end\_date.strftime("%Y-%m-%d"))\

.execute()

# Build daily transaction map (including special events)

daily\_transactions = {}

# Add regular income/expenses

# ... [previous logic] ...

# Add special events

for event in special\_events\_response.data:

date\_key = event['due\_date']

if date\_key not in daily\_transactions:

daily\_transactions[date\_key] = {

'income': 0,

'expenses': 0,

'special\_events': 0,

'events\_count': 0

}

daily\_transactions[date\_key]['special\_events'] += event['amount']

daily\_transactions[date\_key]['events\_count'] += 1

# Calculate daily balances with event tracking

current\_balance = opening\_balance

current\_date = start\_date

daily\_records = []

while current\_date <= end\_date:

date\_str = current\_date.strftime("%Y-%m-%d")

# Get transactions for this date

day\_data = daily\_transactions.get(date\_str, {

'income': 0,

'expenses': 0,

'special\_events': 0,

'events\_count': 0

})

day\_income = day\_data['income']

day\_expenses = day\_data['expenses']

day\_special\_events = day\_data['special\_events']

total\_outflow = day\_expenses + day\_special\_events

net\_change = day\_income - total\_outflow

new\_balance = current\_balance + net\_change

# Determine balance status

if new\_balance < 0:

status = 'critical'

elif new\_balance < 100:

status = 'negative'

else:

status = 'positive'

# Count uncovered/at-risk events for this date

uncovered\_count = 0

at\_risk\_count = 0

if day\_special\_events > 0:

events\_today = [e for e in special\_events\_response.data

if e['due\_date'] == date\_str]

for event in events\_today:

if new\_balance < event['amount']:

uncovered\_count += 1

elif new\_balance < (event['amount'] \* 1.2): # 20% buffer

at\_risk\_count += 1

daily\_record = {

'user\_id': user\_id,

'forecast\_date': date\_str,

'opening\_balance': current\_balance,

'total\_income': day\_income,

'total\_expenses': day\_expenses,

'special\_events\_expense': day\_special\_events,

'closing\_balance': new\_balance,

'net\_daily\_change': net\_change,

'balance\_status': status,

'uncovered\_events\_count': uncovered\_count,

'at\_risk\_events\_count': at\_risk\_count

}

daily\_records.append(daily\_record)

current\_balance = new\_balance

current\_date += timedelta(days=1)

# Batch insert daily records

if daily\_records:

# Clear existing records

self.supabase.table('daily\_cashflow')\

.delete()\

.eq('user\_id', user\_id)\

.gte('forecast\_date', start\_date.strftime("%Y-%m-%d"))\

.execute()

# Insert new records

self.supabase.table('daily\_cashflow').insert(daily\_records).execute()

# Update event coverage status

await self.special\_events.calculate\_event\_coverage(user\_id, days\_ahead)

return True

except Exception as e:

logging.error(f"Error calculating enhanced daily cashflow for user {user\_id}: {e}")

return False

async def refresh\_complete\_forecast(self, user\_id):

"""Complete forecast refresh including special events"""

success = True

# Refresh regular schedules

success &= await self.populate\_income\_schedules(user\_id) # From previous code

success &= await self.populate\_expense\_schedules(user\_id) # From previous code

# Refresh special events

success &= await self.special\_events.populate\_special\_event\_dates(user\_id)

# Calculate enhanced cash flow

success &= await self.calculate\_daily\_cashflow\_with\_events(user\_id)

return success

**Frontend Dashboard Integration**

**Special Events Display Component**

// Special Events Dashboard Component

class SpecialEventsDisplay {

constructor(supabaseClient, userId) {

this.supabase = supabaseClient;

this.userId = userId;

}

async loadSpecialEvents() {

try {

const { data: events, error } = await this.supabase

.from('user\_special\_event\_dates')

.select('\*')

.eq('user\_id', this.userId)

.gte('due\_date', new Date().toISOString().split('T')[0])

.order('due\_date', { ascending: true })

.limit(10);

if (error) throw error;

this.displayEvents(events);

return events;

} catch (error) {

console.error('Error loading special events:', error);

return [];

}

}

displayEvents(events) {

const container = document.getElementById('special-events-container');

if (!container) return;

let html = '<h3>Upcoming Special Events & Milestones</h3>';

if (events.length === 0) {

html += '<p>No upcoming special events. <a href="#" onclick="addSpecialEvent()">Add your first event</a></p>';

} else {

html += '<div class="events-list">';

events.forEach(event => {

const statusClass = this.getStatusClass(event.coverage\_status);

const statusIcon = this.getStatusIcon(event.coverage\_status);

const daysUntil = this.getDaysUntil(event.due\_date);

html += `

<div class="event-card ${statusClass}">

<div class="event-header">

<span class="event-status">${statusIcon}</span>

<h4>${event.event\_name}</h4>

<span class="event-amount">$${event.amount.toLocaleString()}</span>

</div>

<div class="event-details">

<span class="event-date">${this.formatDate(event.due\_date)}</span>

<span class="event-countdown">${daysUntil}</span>

<span class="event-type">${this.formatEventType(event.event\_type)}</span>

</div>

<div class="event-status-text">

${this.getStatusMessage(event.coverage\_status, event.amount)}

</div>

</div>

`;

});

html += '</div>';

}

container.innerHTML = html;

}

getStatusClass(coverageStatus) {

switch(coverageStatus) {

case 'covered': return 'status-green';

case 'at\_risk': return 'status-yellow';

case 'uncovered': return 'status-red';

default: return 'status-gray';

}

}

getStatusIcon(coverageStatus) {

switch(coverageStatus) {

case 'covered': return '✅';

case 'at\_risk': return '⚠️';

case 'uncovered': return '❌';

default: return '❓';

}

}

getStatusMessage(coverageStatus, amount) {

switch(coverageStatus) {

case 'covered':

return `<span class="status-good">✅ Funds available</span>`;

case 'at\_risk':

return `<span class="status-warning">⚠️ Cutting it close - save more!</span>`;

case 'uncovered':

return `<span class="status-danger">❌ Need to save $${amount.toLocaleString()}</span>`;

default:

return 'Status unknown';

}

}

getDaysUntil(dueDate) {

const today = new Date();

const eventDate = new Date(dueDate);

const diffTime = eventDate - today;

const diffDays = Math.ceil(diffTime / (1000 \* 60 \* 60 \* 24));

if (diffDays < 0) return 'Past due';

if (diffDays === 0) return 'Today!';

if (diffDays === 1) return 'Tomorrow';

if (diffDays <= 7) return `${diffDays} days away`;

if (diffDays <= 30) return `${Math.ceil(diffDays / 7)} weeks away`;

return `${Math.ceil(diffDays / 30)} months away`;

}

formatDate(dateString) {

return new Date(dateString).toLocaleDateString('en-US', {

month: 'short',

day: 'numeric',

year: 'numeric'

});

}

formatEventType(eventType) {

const typeMap = {

'child\_birthday': 'Child Birthday',

'spouse\_birthday': 'Spouse Birthday',

'parent\_birthday': 'Parent Birthday',

'anniversary': 'Anniversary',

'solo\_vacation': 'Solo Vacation',

'group\_trip\_girls': 'Girls Trip',

'group\_trip\_boys': 'Boys Trip',

'tax\_refund': 'Tax Refund',

'property\_tax': 'Property Tax',

'car\_registration': 'Car Registration',

'other\_milestone': 'Other'

};

return typeMap[eventType] || eventType;

}

}

// Add Special Event Form

function showAddEventForm() {

const formHtml = `

<div class="modal-overlay" id="add-event-modal">

<div class="modal-card">

<h3>Add Special Event</h3>

<form id="add-event-form">

<div class="form-group">

<label for="event-name">Event Name</label>

<input type="text" id="event-name" required placeholder="e.g., Sarah's Birthday">

</div>

<div class="form-group">

<label for="event-type">Event Type</label>

<select id="event-type" required>

<option value="">Select type...</option>

<option value="child\_birthday">Child's Birthday</option>

<option value="spouse\_birthday">Spouse's Birthday</option>

<option value="parent\_birthday">Parent's Birthday</option>

<option value="anniversary">Anniversary</option>

<option value="solo\_vacation">Solo Vacation</option>

<option value="group\_trip\_girls">Girls Trip</option>

<option value="group\_trip\_boys">Boys Trip</option>

<option value="tax\_refund">Tax Refund</option>

<option value="property\_tax">Property Tax</option>

<option value="car\_registration">Car Registration</option>

<option value="other\_milestone">Other</option>

</select>

</div>

<div class="form-group">

<label for="event-date">Date</label>

<input type="date" id="event-date" required>

</div>

<div class="form-group">

<label for="event-amount">Estimated Amount</label>

<input type="number" id="event-amount" required min="0" step="0.01" placeholder="0.00">

</div>

<div class="form-group">

<label for="priority-level">Priority</label>

<select id="priority-level">

<option value="1">Critical (Must have)</option>

<option value="2">High (Very important)</option>

<option value="3" selected>Medium (Important)</option>

<option value="4">Low (Would be nice)</option>

<option value="5">Optional (If possible)</option>

</select>

</div>

<div class="form-actions">

<button type="button" onclick="closeAddEventModal()">Cancel</button>

<button type="submit">Add Event</button>

</div>

</form>

</div>

</div>

`;

document.body.insertAdjacentHTML('beforeend', formHtml);

document.getElementById('add-event-form').addEventListener('submit', handleAddEvent);

}

async function handleAddEvent(e) {

e.preventDefault();

const eventData = {

user\_id: getCurrentUserId(), // Your user ID function

event\_name: document.getElementById('event-name').value,

event\_type: document.getElementById('event-type').value,

base\_date: document.getElementById('event-date').value,

estimated\_amount: parseFloat(document.getElementById('event-amount').value),

priority\_level: parseInt(document.getElementById('priority-level').value),

frequency: 'annually',

is\_active: true

};

try {

const { data, error } = await supabase

.from('user\_special\_events')

.insert([eventData]);

if (error) throw error;

// Refresh the forecast

await refreshCompleteUserForecast(getCurrentUserId());

// Reload the display

const eventsDisplay = new SpecialEventsDisplay(supabase, getCurrentUserId());

await eventsDisplay.loadSpecialEvents();

// Close modal

closeAddEventModal();

// Show success message

showNotification('Special event added successfully!', 'success');

} catch (error) {

console.error('Error adding special event:', error);

showNotification('Error adding event. Please try again.', 'error');

}

}

**CSS for Status Indicators**

/\* Special Events Styling \*/

.events-list {

display: flex;

flex-direction: column;

gap: 1rem;

}

.event-card {

border: 2px solid #e5e5e5;

border-radius: 8px;

padding: 1rem;

transition: all 0.3s ease;

}

.event-card:hover {

transform: translateY(-2px);

box-shadow: 0 4px 12px rgba(0,0,0,0.1);

}

/\* Status Colors \*/

.status-green {

border-left: 4px solid #10b981;

background-color: #f0fdf4;

}

.status-yellow {

border-left: 4px solid #f59e0b;

background-color: #fffbeb;

}

.status-red {

border-left: 4px solid #ef4444;

background-color: #fef2f2;

}

.event-header {

display: flex;

justify-content: space-between;

align-items: center;

margin-bottom: 0.5rem;

}

.event-header h4 {

margin: 0;

font-size: 1.1rem;

}

.event-amount {

font-weight: bold;

font-size: 1.1rem;

}

.event-details {

display: flex;

gap: 1rem;

font-size: 0.9rem;

color: #666;

margin-bottom: 0.5rem;

}

.event-status-text {

font-size: 0.9rem;

font-weight: 500;

}

.status-good { color: #10b981; }

.status-warning { color: #f59e0b; }

.status-danger { color: #ef4444; }

/\* Modal Styling \*/

.modal-overlay {

position: fixed;

top: 0;

left: 0;

right: 0;

bottom: 0;

background: rgba(0,0,0,0.5);

display: flex;

justify-content: center;

align-items: center;

z-index: 1000;

}

.modal-card {

background: white;

padding: 2rem;

border-radius: 8px;

width: 90%;

max-width: 500px;

max-height: 90vh;

overflow-y: auto;

}

.form-group {

margin-bottom: 1rem;

}

.form-group label {

display: block;

margin-bottom: 0.5rem;

font-weight: 500;

}

.form-group input,

.form-group select {

width: 100%;

padding: 0.5rem;

border: 1px solid #ddd;

border-radius: 4px;

font-size: 1rem;

}

.form-actions {

display: flex;

justify-content: flex-end;

gap: 1rem;

margin-top: 1.5rem;

}

.form-actions button {

padding: 0.5rem 1rem;

border-radius: 4px;

border: none;

cursor: pointer;

}

.form-actions button[type="submit"] {

background: #3b82f6;

color: white;

}

.form-actions button[type="button"] {

background: #6b7280;

color: white;

}

**Workflow Process Summary**

**1. User Adds Special Event**

* User clicks "Add Special Event" button
* Fills out form with event details
* System saves to user\_special\_events table

**2. Date Generation**

* System generates recurring annual dates
* Populates user\_special\_event\_dates table
* Creates entries for next 3 years

**3. Cash Flow Integration**

* Daily cash flow calculation includes special events
* Treats special events as expenses on their due dates
* Calculates if user has enough money for each event

**4. Status Determination**

* **GREEN (Covered)**: User has enough money
* **YELLOW (At Risk)**: User has 80-120% of needed amount
* **RED (Uncovered)**: User doesn't have enough money

**5. Dashboard Display**

* Shows upcoming events with color-coded status
* Displays countdown until event
* Shows amount needed and coverage status

This system gives users powerful milestone planning capabilities while maintaining the visual feedback they need to make informed financial decisions!