**Mingus Goals Setting Screen Development Prompts**

**Overview**

The Goals Setting Screen is a critical missing component in the Mingus onboarding flow (Step 4 of 8). This screen will capture user financial goals and integrate them with the health/wellness tracking and job security features already in development.

**1. Backend Development Prompt**

**Database Schema & API Routes**

**Prompt for Backend Developer:**

Create a comprehensive Goals Setting system for Mingus financial wellness app.

CONTEXT:

- Target users: professionals aged 25-40

- Existing tables: user\_income\_due\_dates, user\_expense\_due\_dates, daily\_cashflow, health\_checkins, relationship\_checkins

- Tech stack: Python/Django backend, Supabase database

- Integration needed with existing cash flow forecasting and special events system

REQUIREMENTS:

1. DATABASE SCHEMA:

Create these tables in Supabase:

```sql

-- Financial goals table

CREATE TABLE user\_financial\_goals (

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

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

goal\_type text CHECK (goal\_type IN (

'emergency\_fund', 'debt\_payoff', 'home\_purchase', 'vacation\_fund',

'wedding\_fund', 'car\_purchase', 'retirement\_savings', 'investment\_portfolio',

'side\_business', 'education\_fund', 'child\_fund', 'custom'

)),

goal\_name text NOT NULL,

target\_amount decimal(12,2) NOT NULL,

current\_amount decimal(12,2) DEFAULT 0.00,

target\_date date,

priority\_level integer CHECK (priority\_level BETWEEN 1 AND 5) DEFAULT 3,

monthly\_contribution decimal(10,2) DEFAULT 0.00,

auto\_contribute boolean DEFAULT false,

status text CHECK (status IN ('active', 'paused', 'completed', 'cancelled')) DEFAULT 'active',

motivation\_note text,

milestone\_amounts decimal[],

created\_at timestamp DEFAULT now(),

updated\_at timestamp DEFAULT now()

);

-- Goal progress tracking

CREATE TABLE goal\_progress\_tracking (

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

goal\_id uuid REFERENCES user\_financial\_goals(id) NOT NULL,

progress\_date date NOT NULL,

amount\_contributed decimal(10,2) NOT NULL,

balance\_after\_contribution decimal(12,2) NOT NULL,

contribution\_source text, -- 'manual', 'automatic', 'windfall'

notes text,

created\_at timestamp DEFAULT now()

);

-- Goal-health correlation tracking

CREATE TABLE goal\_health\_correlations (

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

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

goal\_id uuid REFERENCES user\_financial\_goals(id) NOT NULL,

week\_date date NOT NULL,

stress\_level integer,

progress\_satisfaction integer CHECK (progress\_satisfaction BETWEEN 1 AND 10),

motivation\_level integer CHECK (motivation\_level BETWEEN 1 AND 10),

goal\_related\_spending decimal(10,2) DEFAULT 0.00,

created\_at timestamp DEFAULT now()

);

1. API ENDPOINTS: Create Django routes in backend/routes/goals.py:

@goals\_bp.route('/setup', methods=['GET', 'POST'])

def setup\_goals():

"""Initial goals setup during onboarding"""

@goals\_bp.route('/dashboard', methods=['GET'])

def goals\_dashboard():

"""Get user's goals overview for dashboard"""

@goals\_bp.route('/add', methods=['POST'])

def add\_goal():

"""Add new financial goal"""

@goals\_bp.route('/<goal\_id>/update', methods=['PUT'])

def update\_goal():

"""Update existing goal"""

@goals\_bp.route('/<goal\_id>/contribute', methods=['POST'])

def record\_contribution():

"""Record progress toward goal"""

@goals\_bp.route('/progress-analysis', methods=['GET'])

def analyze\_goal\_progress():

"""Analyze goal progress vs health metrics"""

1. GOAL CALCULATION LOGIC: Implement these functions:

def calculate\_monthly\_contribution\_needed(target\_amount, current\_amount, target\_date):

"""Calculate required monthly contribution to reach goal"""

def analyze\_goal\_feasibility(user\_income, user\_expenses, goal\_amount, timeframe):

"""Determine if goal is realistic given current finances"""

def suggest\_goal\_adjustments(user\_data, goal\_data):

"""Suggest timeline or amount adjustments for unrealistic goals"""

def calculate\_goal\_stress\_impact(goal\_progress, user\_stress\_levels):

"""Correlate goal progress with user stress from health check-ins"""

1. INTEGRATION REQUIREMENTS:

* Connect with existing daily\_cashflow table to show goal impact
* Integrate with health\_checkins to track motivation/stress correlation
* Link with special\_events for goal-related celebrations
* Prepare for job security integration (adjust goals based on employment risk)

DELIVERABLES:

* Complete database migration files
* Fully functional API endpoints with proper error handling
* Goal calculation algorithms with edge case handling
* Integration with existing cash flow forecasting
* Unit tests for all goal calculation functions

---

## 2. Frontend Development Prompt

### React Component & UI Development

\*\*Prompt for Frontend Developer:\*\*

Build the Goals Setting Screen React component for Mingus financial wellness app onboarding.

CONTEXT:

* Step 4 of 8 in onboarding flow
* Target users: Young African American professionals
* Modern, engaging UI that feels premium and culturally relevant
* Integration with existing health tracking and cash flow features
* Tech stack: React with hooks, Tailwind CSS only

REQUIREMENTS:

1. MAIN COMPONENT STRUCTURE: Create GoalsSetup.tsx with these sections:

interface Goal {

id?: string;

type: GoalType;

name: string;

targetAmount: number;

currentAmount: number;

targetDate: Date;

priority: 1 | 2 | 3 | 4 | 5;

monthlyContribution: number;

motivationNote: string;

}

type GoalType = 'emergency\_fund' | 'debt\_payoff' | 'home\_purchase' | 'vacation\_fund' |

'wedding\_fund' | 'car\_purchase' | 'retirement\_savings' | 'investment\_portfolio' |

'side\_business' | 'education\_fund' | 'child\_fund' | 'important\_dates';

1. UI SECTIONS:

A. WELCOME/INTRO SECTION:

<div className="goals-intro">

<h1>Let's Set Your Financial Goals 🎯</h1>

<p>Your goals drive everything else. Let's map out what you're working toward.</p>

<div className="cultural-connection">

<p>"A goal without a plan is just a wish. Let's make yours reality." 💪</p>

</div>

</div>

B. GOAL TYPE SELECTOR:

<div className="goal-types-grid">

{goalTypes.map(type => (

<GoalTypeCard

key={type.id}

icon={type.icon}

name={type.name}

description={type.description}

examples={type.examples}

selected={selectedGoals.includes(type.id)}

onClick={() => toggleGoalSelection(type.id)}

/>

))}

</div>

Goal types with cultural relevance:

* 🏠 Home Ownership ("Build generational wealth")
* 💒 Wedding Fund ("Celebrate your love properly")
* ✈️ Travel Fund ("See the world, embrace experiences")
* 🚗 Reliable Transportation ("Independence and mobility")
* 📚 Education Investment ("Keep learning, keep growing")
* 👶 Family Planning ("Secure the next generation")
* 💼 Side Business ("Build multiple income streams")
* 🛡️ Emergency Fund ("Protect your peace of mind")

C. GOAL DETAILS FORM:

<div className="goal-details-form">

<div className="amount-input-section">

<label>How much do you need?</label>

<div className="amount-input-wrapper">

<span className="currency-symbol">$</span>

<input

type="number"

value={goalAmount}

onChange={handleAmountChange}

placeholder="0"

className="amount-input"

/>

</div>

<div className="amount-suggestions">

{/\* Dynamic suggestions based on goal type \*/}

</div>

</div>

<div className="timeline-section">

<label>When do you want to achieve this?</label>

<div className="timeline-options">

<button onClick={() => setTimeline('6months')}>6 months</button>

<button onClick={() => setTimeline('1year')}>1 year</button>

<button onClick={() => setTimeline('2years')}>2 years</button>

<button onClick={() => setTimeline('5years')}>5+ years</button>

<input type="date" onChange={handleCustomDate} />

</div>

</div>

<div className="contribution-analysis">

<div className="monthly-needed">

<h3>Monthly Contribution Needed</h3>

<div className="amount-display">${monthlyNeeded}</div>

<div className="feasibility-indicator">

{feasible ? "✅ Achievable" : "⚠️ Stretch Goal"}

</div>

</div>

</div>

<div className="motivation-section">

<label>What's driving this goal? (Optional but powerful)</label>

<textarea

placeholder="e.g., 'Want to buy a house in my childhood neighborhood' or 'Building security for my family'"

value={motivation}

onChange={setMotivation}

className="motivation-input"

/>

</div>

</div>

D. PROGRESS VISUALIZATION:

<div className="goal-progress-preview">

<h3>Your Progress Roadmap</h3>

<div className="progress-timeline">

{milestones.map((milestone, index) => (

<div key={index} className="milestone-point">

<div className="milestone-date">{milestone.date}</div>

<div className="milestone-amount">${milestone.amount}</div>

<div className="milestone-celebration">{milestone.celebration}</div>

</div>

))}

</div>

</div>

1. INTERACTIVE FEATURES:

A. SMART SUGGESTIONS:

const SmartSuggestions = ({ userIncome, userExpenses, goalType }) => {

const suggestions = generateGoalSuggestions(userIncome, userExpenses, goalType);

return (

<div className="smart-suggestions">

<h4>💡 Based on your income, consider:</h4>

{suggestions.map(suggestion => (

<button

key={suggestion.amount}

onClick={() => applyAmount(suggestion.amount)}

className="suggestion-button"

>

${suggestion.amount.toLocaleString()} - {suggestion.reasoning}

</button>

))}

</div>

);

};

B. FEASIBILITY CHECKER:

const FeasibilityChecker = ({ goal, userFinances }) => {

const analysis = analyzeFeasibility(goal, userFinances);

return (

<div className={`feasibility-card ${analysis.level}`}>

<div className="feasibility-icon">{analysis.icon}</div>

<div className="feasibility-message">{analysis.message}</div>

{analysis.adjustments && (

<div className="adjustment-suggestions">

<h5>Consider these adjustments:</h5>

<ul>

{analysis.adjustments.map(adj => (

<li key={adj.type}>

<button onClick={() => applyAdjustment(adj)}>

{adj.description}

</button>

</li>

))}

</ul>

</div>

)}

</div>

);

};

1. STYLING REQUIREMENTS:

Use Tailwind classes for:

* Vibrant, engaging color scheme (blues, greens, golds)
* Smooth animations and micro-interactions
* Mobile-first responsive design
* Cultural elements (appropriate imagery, language)
* Progress indicators and visual feedback
* Accessibility compliance

1. STATE MANAGEMENT:

const useGoalsSetup = () => {

const [selectedGoals, setSelectedGoals] = useState([]);

const [currentGoalIndex, setCurrentGoalIndex] = useState(0);

const [goals, setGoals] = useState([]);

const [userFinances, setUserFinances] = useState(null);

// All state management logic

};

1. INTEGRATION POINTS:

* Pull user income/expense data from previous onboarding steps
* Save goals to backend API endpoints
* Navigate to next onboarding step (Health Assessment)
* Handle error states gracefully
* Provide preview of how goals integrate with dashboard

DELIVERABLES:

* Complete GoalsSetup.tsx component
* Supporting components (GoalTypeCard, FeasibilityChecker, etc.)
* Responsive styling with Tailwind
* Integration with onboarding flow
* Error handling and loading states
* Accessibility features

---

## 3. UI/UX Design Prompt

### Visual Design & User Experience

\*\*Prompt for UI/UX Designer:\*\*

Design the Goals Setting Screen for Mingus, a financial wellness app targeting professionals aged 25-40.

DESIGN BRIEF: This is Step 4 of an 8-step onboarding flow. Users have already provided basic profile and financial information. Now we need to capture their financial goals in a way that feels inspiring, achievable, and culturally relevant.

DESIGN REQUIREMENTS:

1. OVERALL AESTHETIC:

* Modern, premium feel that builds trust
* Vibrant but professional color palette
* Cultural relevance without stereotypes
* Mobile-first design (primary usage expected on phones)
* Engaging micro-animations and interactions

1. COLOR PALETTE: Primary: Deep navy (#1a1a2e) and rich blue (#16213e) Secondary: Bright cyan (#00d9ff) for accents and CTAs Success: Forest green (#4caf50) for positive indicators Warning: Warm orange (#ffa726) for stretch goals Danger: Modern red (#ff4757) for unrealistic goals Neutral: Warm grays (#f8f9fa, #404040)
2. SCREEN SECTIONS TO DESIGN:

A. WELCOME SECTION:

┌─────────────────────────────────────┐

│ [Progress: 4/8 ●●●●○○○○] │

│ │

│ 🎯 Let's Set Your Financial Goals │

│ │

│ Your goals are the foundation of │

│ your financial wellness journey. │

│ Let's map out what matters most. │

│ │

│ 💪 "A goal without a plan is just │

│ a wish. Let's make yours real." │

└─────────────────────────────────────┘

B. GOAL TYPE SELECTION GRID: Design cards for each goal type with:

* Relevant emoji/icon
* Goal name
* Brief inspiring description
* Example amounts
* Selection state (border highlight, checkmark)

Goal types to design: 🏠 Home Ownership - "Build generational wealth" 💒 Wedding Fund - "Celebrate your love story" ✈️ Travel & Experiences - "Invest in memories" 🚗 Reliable Transportation - "Freedom and independence" 📚 Education Investment - "Level up your expertise" 👶 Family Planning - "Secure the future" 💼 Side Business - "Multiple income streams" 🛡️ Emergency Fund - "Peace of mind protection"

C. GOAL DETAILS FORM: Design interactive form with:

* Large, prominent amount input with currency formatting
* Smart suggestion buttons below amount input
* Timeline selector (quick options + custom date)
* Monthly contribution calculator with visual feedback
* Feasibility indicator (green check, yellow warning, red flag)
* Motivation text area with inspiring prompt

D. PROGRESS VISUALIZATION: Design a roadmap showing:

* Timeline with milestone markers
* Celebration moments at key milestones
* Visual progress indicators
* Connection to special events (birthdays, anniversaries)

1. INTERACTION DESIGN:

A. MICRO-ANIMATIONS:

* Card selection: Gentle scale + border glow
* Amount input: Number roll animation for calculations
* Feasibility changes: Smooth color transitions
* Progress updates: Fill animations

B. FEEDBACK PATTERNS:

* Success states: Green checkmarks with subtle pulse
* Warning states: Orange indicators with gentle shake
* Error states: Red borders with helpful messaging
* Loading states: Skeleton screens, not spinners

C. GESTURE SUPPORT:

* Swipe between goal setup steps
* Pull-to-refresh for recalculations
* Tap and hold for additional options

1. MOBILE-SPECIFIC CONSIDERATIONS:

* Thumb-friendly touch targets (minimum 44px)
* Single-column layout with clear visual hierarchy
* Sticky progress bar at top
* Floating action button for "Continue"
* Keyboard-aware input handling

1. ACCESSIBILITY FEATURES:

* High contrast ratios (minimum 4.5:1)
* Screen reader friendly labels
* Touch target size compliance
* Focus indicators for keyboard navigation
* Alternative text for all visual elements

1. CULTURAL DESIGN ELEMENTS:

* Authentic representation in any imagery
* Language that resonates with target demographic
* Goal examples relevant to community values
* Success stories and motivational quotes
* Celebration moments that feel meaningful

1. RESPONSIVE BREAKPOINTS:

* Mobile: 320px - 768px (primary focus)
* Tablet: 768px - 1024px
* Desktop: 1024px+ (secondary support)

DELIVERABLES:

* High-fidelity mockups for all screen states
* Interactive prototype showing key user flows
* Component library for goal-related UI elements
* Responsive design specifications
* Accessibility annotation document
* Animation specification guide
* Style guide updates for goal-related components

---

## 4. Integration Prompt

### System Integration & Data Flow

\*\*Prompt for Full-Stack Developer:\*\*

Integrate the Goals Setting Screen into the existing Mingus onboarding flow and connect it with the cash flow forecasting, health tracking, and special events systems.

INTEGRATION REQUIREMENTS:

1. ONBOARDING FLOW INTEGRATION:

A. Navigation Logic:

# Update onboarding progress tracking

def update\_onboarding\_progress(user\_id, completed\_step):

"""Track user progress through onboarding"""

progress\_data = {

'user\_id': user\_id,

'current\_step': 'goals\_setup',

'completed\_steps': ['welcome', 'profile\_setup', 'financial\_profile', 'goals\_setup'],

'progress\_percentage': 50, # 4/8 steps complete

'next\_step': 'health\_assessment'

}

# Update database and return next step URL

# Goals completion validation

def validate\_goals\_completion(user\_goals):

"""Ensure user has set at least one realistic goal"""

if not user\_goals:

return {'valid': False, 'message': 'At least one goal is required'}

realistic\_goals = [g for g in user\_goals if g.feasibility != 'unrealistic']

if not realistic\_goals:

return {'valid': False, 'message': 'Please adjust at least one goal to be achievable'}

return {'valid': True}

B. Data Persistence:

# Save goals during onboarding

@onboarding\_bp.route('/goals-setup', methods=['POST'])

def save\_onboarding\_goals():

"""Save user goals and calculate integration impacts"""

user\_goals = request.json.get('goals', [])

user\_id = get\_current\_user\_id()

# Validate goals

validation = validate\_goals\_completion(user\_goals)

if not validation['valid']:

return jsonify({'error': validation['message']}), 400

# Save goals to database

saved\_goals = []

for goal\_data in user\_goals:

goal = create\_user\_goal(user\_id, goal\_data)

saved\_goals.append(goal)

# Update cash flow projections with goal contributions

update\_cashflow\_with\_goals(user\_id, saved\_goals)

# Create initial goal milestones as special events

create\_goal\_milestone\_events(user\_id, saved\_goals)

# Update onboarding progress

update\_onboarding\_progress(user\_id, 'goals\_setup')

return jsonify({

'success': True,

'goals\_saved': len(saved\_goals),

'next\_step': '/onboarding/health-assessment',

'cashflow\_impact': calculate\_cashflow\_impact(saved\_goals)

})

1. CASH FLOW INTEGRATION:

A. Goal Impact on Daily Cash Flow:

def integrate\_goals\_into\_cashflow(user\_id, days\_ahead=365):

"""Enhance daily cash flow calculation to include goal contributions"""

# Get user's active goals

active\_goals = get\_user\_active\_goals(user\_id)

# Calculate daily goal allocations

daily\_goal\_contributions = {}

for goal in active\_goals:

contribution\_schedule = calculate\_goal\_contribution\_schedule(goal)

for date, amount in contribution\_schedule.items():

if date not in daily\_goal\_contributions:

daily\_goal\_contributions[date] = 0

daily\_goal\_contributions[date] += amount

# Modify existing daily\_cashflow calculation

def calculate\_enhanced\_daily\_cashflow\_with\_goals(user\_id, days\_ahead):

# ... existing cash flow logic ...

for day in range(days\_ahead):

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

# ... existing income/expense calculations ...

# Add goal contributions as planned expenses

day\_goal\_contributions = daily\_goal\_contributions.get(date\_str, 0)

total\_outflow = day\_expenses + day\_events + day\_goal\_contributions

# ... rest of calculation ...

daily\_record.update({

'goal\_contributions': day\_goal\_contributions,

'goal\_impact\_analysis': analyze\_goal\_impact\_on\_cashflow(

closing\_balance, day\_goal\_contributions, active\_goals

)

})

return enhanced\_cashflow\_data

def analyze\_goal\_impact\_on\_cashflow(closing\_balance, goal\_contributions, goals):

"""Analyze how goal contributions affect financial health"""

if goal\_contributions == 0:

return {'status': 'no\_impact'}

balance\_after\_goals = closing\_balance - goal\_contributions

if balance\_after\_goals < 0:

# Find which goals to potentially pause

pausable\_goals = [g for g in goals if g.priority <= 3]

return {

'status': 'negative\_impact',

'suggested\_action': 'pause\_low\_priority\_goals',

'pausable\_goals': [g.id for g in pausable\_goals]

}

elif balance\_after\_goals < 500: # Low balance threshold

return {

'status': 'tight\_cashflow',

'suggested\_action': 'review\_goal\_timing'

}

else:

return {'status': 'healthy\_impact'}

B. Goal Progress Tracking Integration:

def update\_goal\_progress\_from\_transactions(user\_id):

"""Automatically update goal progress from actual transactions"""

# Get recent transactions (from bank integration)

recent\_transactions = get\_user\_transactions(user\_id, days=7)

# Get user's active goals

active\_goals = get\_user\_active\_goals(user\_id)

for goal in active\_goals:

# Look for transactions that match goal contribution patterns

goal\_transactions = identify\_goal\_contributions(goal, recent\_transactions)

for transaction in goal\_transactions:

record\_goal\_contribution(

goal.id,

transaction.amount,

transaction.date,

source='automatic\_detection'

)

# Check if goal is completed

if goal.current\_amount >= goal.target\_amount:

complete\_goal(goal.id)

create\_goal\_completion\_celebration(goal)

1. HEALTH TRACKING INTEGRATION:

A. Goal-Stress Correlation:

def correlate\_goal\_progress\_with\_health(user\_id):

"""Analyze relationship between goal progress and health metrics"""

# Get recent health check-ins

health\_data = get\_recent\_health\_checkins(user\_id, weeks=4)

# Get goal progress over same period

goal\_progress = get\_goal\_progress\_history(user\_id, weeks=4)

correlations = []

for week\_data in health\_data:

week\_goals = get\_goals\_for\_week(goal\_progress, week\_data.week)

correlation = {

'week': week\_data.week,

'stress\_level': week\_data.stress\_level,

'goal\_progress\_satisfaction': calculate\_goal\_satisfaction(week\_goals),

'goals\_behind\_schedule': count\_behind\_schedule\_goals(week\_goals),

'spending\_correlation': analyze\_stress\_spending\_vs\_goals(

week\_data, week\_goals

)

}

correlations.append(correlation)

# Generate insights

insights = generate\_goal\_health\_insights(correlations)

return insights

def generate\_goal\_health\_insights(correlations):

"""Generate personalized insights about goal progress and health"""

insights = []

# Check for stress-spending patterns affecting goals

high\_stress\_weeks = [c for c in correlations if c['stress\_level'] > 7]

if high\_stress\_weeks:

avg\_goal\_progress = sum(w['goal\_progress\_satisfaction'] for w in high\_stress\_weeks) / len(high\_stress\_weeks)

if avg\_goal\_progress < 5:

insights.append({

'type': 'stress\_goal\_impact',

'message': 'High stress weeks tend to slow your goal progress. Consider stress management techniques.',

'recommendation': 'Build small stress-relief activities into your budget.'

})

return insights

1. SPECIAL EVENTS INTEGRATION:

A. Goal Milestones as Special Events:

def create\_goal\_milestone\_events(user\_id, goals):

"""Create special events for goal milestones and celebrations"""

for goal in goals:

# Calculate milestone dates (25%, 50%, 75%, 100%)

milestones = calculate\_goal\_milestones(goal)

for milestone in milestones:

# Create celebration event

celebration\_event = {

'user\_id': user\_id,

'event\_name': f'{goal.name} - {milestone.percentage}% Complete! 🎉',

'event\_type': 'goal\_milestone',

'event\_date': milestone.projected\_date,

'estimated\_amount': milestone.celebration\_budget,

'priority\_level': 4, # High priority for celebration

'goal\_id': goal.id,

'milestone\_percentage': milestone.percentage

}

create\_special\_event(celebration\_event)

# Create goal completion celebration

completion\_event = {

'user\_id': user\_id,

'event\_name': f'{goal.name} Achievement Celebration! 🏆',

'event\_type': 'goal\_completion',

'event\_date': goal.target\_date,

'estimated\_amount': calculate\_celebration\_budget(goal.target\_amount),

'priority\_level': 5, # Maximum priority

'goal\_id': goal.id

}

create\_special\_event(completion\_event)

def calculate\_celebration\_budget(goal\_amount):

"""Calculate appropriate celebration budget based on goal size"""

if goal\_amount < 1000:

return 50 # Small celebration

elif goal\_amount < 10000:

return goal\_amount \* 0.02 # 2% of goal

else:

return min(goal\_amount \* 0.01, 500) # 1% up to $500 max

1. DASHBOARD INTEGRATION:

A. Goals Widget for Main Dashboard:

def get\_goals\_dashboard\_data(user\_id):

"""Get goal data formatted for dashboard display"""

active\_goals = get\_user\_active\_goals(user\_id)

dashboard\_data = {

'total\_goals': len(active\_goals),

'total\_target\_amount': sum(g.target\_amount for g in active\_goals),

'total\_saved': sum(g.current\_amount for g in active\_goals),

'overall\_progress\_percentage': calculate\_overall\_progress(active\_goals),

'next\_milestone': get\_next\_milestone(active\_goals),

'monthly\_contribution\_total': sum(g.monthly\_contribution for g in active\_goals),

'goals\_on\_track': count\_goals\_on\_track(active\_goals),

'goals\_behind': count\_goals\_behind\_schedule(active\_goals),

'recent\_achievements': get\_recent\_goal\_achievements(user\_id, days=30)

}

return dashboard\_data

INTEGRATION TESTING REQUIREMENTS:

* Test onboarding flow with goals data persistence
* Verify cash flow calculations include goal contributions
* Test goal-health correlation calculations
* Verify special events creation for goal milestones
* Test goal progress updates from transaction data
* Validate dashboard goal widget displays correctly

DELIVERABLES:

* Complete integration between goals and existing systems
* Enhanced daily cash flow calculation with goals
* Goal milestone special events creation
* Goal-health correlation analysis
* Dashboard goals widget
* Comprehensive integration tests

---

## Summary

These prompts provide comprehensive direction for implementing the Goals Setting Screen across all development disciplines. The key integration points ensure the goals feature enhances rather than complicates the existing Mingus ecosystem, while maintaining the cultural relevance and user experience quality your target demographic expects.

Each prompt is designed to be actionable and specific enough for developers to implement independently while ensuring consistency across the full system.

**Mingus Goals Setting Screen - Optimized Development Sequence**

**Overview**

This document outlines the optimal order for executing the Goals Setting Screen development prompts to minimize errors, reduce dependencies, and ensure smooth integration with existing Mingus systems.

**Development Phase Structure**

**Phase 1: Foundation & Planning (Days 1-2)**

*Build the foundation before any code is written*

**Phase 2: Backend Core (Days 3-5)**

*Establish data layer and business logic*

**Phase 3: Integration Layer (Days 6-8)**

*Connect with existing systems*

**Phase 4: Frontend Development (Days 9-12)**

*Build user interface components*

**Phase 5: Testing & Refinement (Days 13-15)**

*Comprehensive testing and optimization*

**Detailed Execution Sequence**

**PHASE 1: Foundation & Planning (Days 1-2)**

**Step 1.1: UI/UX Design First ⭐ CRITICAL START POINT**

**Execute Prompt:** UI/UX Design Prompt **Why First:**

* Defines the complete user experience before code constraints
* Identifies all data requirements and user flows
* Prevents UI/backend misalignment issues
* Establishes component structure for frontend development

**Deliverables Required Before Moving Forward:**

* [ ] Complete wireframes and mockups
* [ ] Component library specifications
* [ ] Data flow diagrams showing user interactions
* [ ] Responsive design specifications
* [ ] Accessibility requirements documentation

**Dependencies:** None **Risk Level:** Low **Estimated Time:** 2 days

**Step 1.2: Technical Architecture Review**

**Execute:** Internal review of design against existing Mingus architecture **Focus Areas:**

* Database schema compatibility with existing tables
* API endpoint naming consistency
* Frontend component integration points
* Performance impact assessment

**Deliverables:**

* [ ] Architecture compatibility report
* [ ] Identified potential conflicts with existing systems
* [ ] Resource allocation plan
* [ ] Risk mitigation strategies

**PHASE 2: Backend Core Development (Days 3-5)**

**Step 2.1: Database Schema Implementation ⭐ CRITICAL FOUNDATION**

**Execute Prompt:** Backend Development Prompt (Database Schema Section Only) **Why Second:**

* All other development depends on stable data structures
* Schema changes become exponentially more difficult later
* Enables parallel frontend development once APIs are defined

**Specific Focus:**

-- Execute in this exact order:

1. user\_financial\_goals table

2. goal\_progress\_tracking table

3. goal\_health\_correlations table

4. Database indexes for performance

5. Initial data validation triggers

**Validation Required:**

* [ ] Schema successfully created in Supabase
* [ ] All foreign key relationships working
* [ ] Database constraints functioning correctly
* [ ] Sample data insertion/retrieval working
* [ ] Performance baseline established

**Dependencies:** Design Phase complete **Risk Level:** High (foundation for everything else) **Estimated Time:** 1 day

**Step 2.2: Core Goal Calculation Logic**

**Execute Prompt:** Backend Development Prompt (Goal Calculation Functions) **Why Third:**

* Pure business logic - no external dependencies
* Can be unit tested in isolation
* Required for API endpoint functionality

**Specific Implementation Order:**

1. calculate\_monthly\_contribution\_needed()

2. analyze\_goal\_feasibility()

3. suggest\_goal\_adjustments()

4. calculate\_goal\_stress\_impact()

**Validation Required:**

* [ ] All calculation functions pass unit tests
* [ ] Edge cases handled (zero amounts, past dates, etc.)
* [ ] Performance benchmarks met
* [ ] Documentation complete

**Dependencies:** Database schema complete **Risk Level:** Medium **Estimated Time:** 1.5 days

**Step 2.3: Basic API Endpoints**

**Execute Prompt:** Backend Development Prompt (API Endpoints Section) **Why Fourth:**

* Enables frontend development to begin
* Provides testable interface for business logic
* Isolates integration complexity

**Implementation Order:**

1. POST /api/goals/setup (basic goal creation)

2. GET /api/goals/dashboard (goal retrieval)

3. PUT /api/goals/<id>/update (goal modification)

4. POST /api/goals/<id>/contribute (progress tracking)

5. GET /api/goals/progress-analysis (advanced analytics)

**Validation Required:**

* [ ] All endpoints return expected data structures
* [ ] Error handling implemented for all failure modes
* [ ] API documentation generated
* [ ] Postman/API tests passing

**Dependencies:** Core calculations complete **Risk Level:** Medium **Estimated Time:** 1.5 days

**PHASE 3: Integration Layer (Days 6-8)**

**Step 3.1: Cash Flow Integration ⭐ HIGHEST COMPLEXITY**

**Execute Prompt:** Integration Prompt (Cash Flow Integration Section) **Why Fifth:**

* Most complex integration with highest error potential
* Affects core app functionality
* Requires thorough testing before UI development

**Specific Focus:**

1. integrate\_goals\_into\_cashflow() - Core modification

2. analyze\_goal\_impact\_on\_cashflow() - Impact analysis

3. update\_goal\_progress\_from\_transactions() - Automation

**Critical Testing:**

* [ ] Existing cash flow calculations remain accurate
* [ ] Goal contributions properly integrated
* [ ] Performance impact within acceptable bounds
* [ ] Edge cases handled (negative balances, goal pausing)

**Dependencies:** Basic API endpoints functional **Risk Level:** Very High (touches core app functionality) **Estimated Time:** 2 days

**Step 3.2: Health Tracking Integration**

**Execute Prompt:** Integration Prompt (Health Tracking Integration Section) **Why Sixth:**

* Lower risk than cash flow integration
* Builds on existing health check-in system
* Provides valuable user insights

**Implementation Order:**

1. correlate\_goal\_progress\_with\_health()

2. generate\_goal\_health\_insights()

3. Update health check-in UI to include goal satisfaction questions

**Dependencies:** Cash flow integration stable **Risk Level:** Medium **Estimated Time:** 1 day

**Step 3.3: Special Events Integration**

**Execute Prompt:** Integration Prompt (Special Events Integration Section) **Why Seventh:**

* Enhances user experience without affecting core functionality
* Lower risk integration
* Can be delayed if other issues arise

**Dependencies:** Health integration complete **Risk Level:** Low **Estimated Time:** 1 day

**PHASE 4: Frontend Development (Days 9-12)**

**Step 4.1: Core React Components**

**Execute Prompt:** Frontend Development Prompt (Component Structure Section) **Why Eighth:**

* Backend API is stable and tested
* Design specifications are complete
* Reduces frontend/backend iteration cycles

**Implementation Order:**

1. GoalTypeCard component (simplest, no API calls)

2. GoalsSetup main component structure

3. Goal details form components

4. Smart suggestions and feasibility checker

5. Progress visualization components

**Dependencies:** All backend integrations complete **Risk Level:** Low to Medium **Estimated Time:** 2 days

**Step 4.2: API Integration & State Management**

**Execute Prompt:** Frontend Development Prompt (Integration Points Section) **Why Ninth:**

* Components exist for testing
* Backend is stable
* Allows for comprehensive user flow testing

**Dependencies:** Core components complete **Risk Level:** Medium **Estimated Time:** 1 day

**Step 4.3: Onboarding Flow Integration**

**Execute Prompt:** Frontend Development Prompt (Onboarding Flow Integration) **Why Tenth:**

* Requires both frontend and backend to be functional
* Final piece before user testing

**Dependencies:** API integration complete **Risk Level:** Low **Estimated Time:** 1 day

**PHASE 5: Testing & Refinement (Days 13-15)**

**Step 5.1: Integration Testing**

**Focus:** End-to-end user flows

* [ ] Complete onboarding including goals setup
* [ ] Goal creation and modification
* [ ] Cash flow impact verification
* [ ] Health correlation functionality

**Step 5.2: Performance Testing**

**Focus:** System performance with goals integration

* [ ] Database query performance
* [ ] Cash flow calculation speed
* [ ] Frontend rendering performance

**Step 5.3: User Acceptance Testing**

**Focus:** Real user interactions

* [ ] Onboarding flow usability
* [ ] Goal setting experience
* [ ] Cultural relevance validation

**Critical Dependencies & Risk Mitigation**

**Hard Dependencies (Cannot Parallel Process)**

Design → Database Schema → API Endpoints → Integration → Frontend

**Parallel Processing Opportunities**

* Unit tests can be written alongside each component
* Documentation can be created during development
* Performance monitoring setup can occur early

**High-Risk Transition Points**

1. **Design → Database Schema:** Misalignment can cause major rework
2. **API Development → Cash Flow Integration:** Core system modification risk
3. **Backend Complete → Frontend Start:** API contract changes become expensive

**Risk Mitigation Strategies**

**For Database Schema Phase:**

* Create comprehensive test data before proceeding
* Implement rollback procedures
* Test schema with realistic data volumes

**For Cash Flow Integration:**

* Implement feature flags for gradual rollout
* Maintain parallel calculation paths during transition
* Create comprehensive backup/restore procedures

**For Frontend Development:**

* Mock API endpoints early for parallel development
* Implement progressive enhancement approach
* Use component-driven development methodology

**Quality Gates & Go/No-Go Criteria**

**Phase 1 → Phase 2 Criteria:**

* [ ] All design mockups approved
* [ ] Technical architecture review complete
* [ ] Resource allocation confirmed

**Phase 2 → Phase 3 Criteria:**

* [ ] Database schema validated with test data
* [ ] Core calculations pass all unit tests
* [ ] Basic API endpoints functional and documented

**Phase 3 → Phase 4 Criteria:**

* [ ] Cash flow integration tested and stable
* [ ] All backend integrations complete
* [ ] Performance benchmarks met

**Phase 4 → Phase 5 Criteria:**

* [ ] Complete user flow functional
* [ ] Frontend components responsive and accessible
* [ ] Integration with onboarding flow working

**Phase 5 → Production Criteria:**

* [ ] All integration tests passing
* [ ] Performance requirements met
* [ ] User acceptance testing complete
* [ ] Documentation and training materials ready

**Resource Allocation Recommendations**

**Team Composition by Phase:**

**Phase 1:** UI/UX Designer (primary), Technical Lead (review) **Phase 2:** Backend Developer (primary), Database Administrator (support) **Phase 3:** Senior Full-Stack Developer (primary), Backend Developer (support) **Phase 4:**Frontend Developer (primary), UI/UX Designer (support) **Phase 5:** QA Engineer (primary), All team members (support)

**Communication Schedule:**

* Daily standups during active development phases
* Phase transition meetings with full team
* Weekly stakeholder updates on progress and blockers

**Contingency Planning**

**If Database Schema Issues Arise:**

* Fallback to simpler schema design
* Implement goals as JSON objects initially
* Plan proper normalization for future sprint

**If Cash Flow Integration Too Complex:**

* Implement goals as separate tracking initially
* Plan integration for future release
* Focus on user experience over technical perfection

**If Frontend Development Behind Schedule:**

* Prioritize core goal setting functionality
* Delay advanced features (smart suggestions, correlations)
* Implement progressive enhancement approach

**Success Metrics**

**Technical Metrics:**

* Database query performance < 100ms
* Cash flow calculation time increase < 20%
* Frontend component render time < 50ms
* API response time < 200ms

**User Experience Metrics:**

* Goal setup completion rate > 85%
* Onboarding drop-off at goals step < 10%
* User satisfaction with goal features > 4.5/5
* Time to complete goal setup < 5 minutes

**Integration Quality Metrics:**

* Zero critical bugs in cash flow calculations
* Health correlation accuracy > 90%
* Special events creation success rate 100%
* Dashboard goal widget load time < 1 second

**Cursor Responses:**

**Implementation Detail Requests for Goals Setting Backend**

**Response to Developer Questions**

Based on the development sequence plan, you should request **ALL** of the following to maintain the optimized development flow:

**1. Full DB Access Logic Implementation ⭐ PRIORITY 1**

**Request:** "Yes, please provide the complete service-layer DB logic implementation."

**Why Critical Now:**

* You're at Step 2.1 (Database Schema) → Step 2.2 (Core Logic) transition
* Service layer must be solid before building API endpoints
* Prevents API redesign when database interactions prove complex

**Specific Request:**

Please provide complete service-layer implementation including:

1. CRUD Operations for all goal-related tables:

- user\_financial\_goals (create, read, update, delete, list)

- goal\_progress\_tracking (create, read, list with date ranges)

- goal\_health\_correlations (create, read, analytics queries)

2. Complex Query Logic:

- Get goals with progress calculations

- Get goals filtered by status/priority

- Get goal progress history with date ranges

- Get goal-health correlation analytics

- Get goal impact on cash flow projections

3. Data Validation Logic:

- Goal amount validation (realistic ranges)

- Date validation (target dates in future)

- Progress validation (contributions don't exceed targets)

- User ownership validation (security)

4. Business Logic Methods:

- calculate\_goal\_completion\_percentage()

- determine\_goal\_status() (on\_track, behind, ahead)

- calculate\_required\_monthly\_contribution()

- analyze\_goal\_feasibility\_with\_user\_finances()

**2. Django ORM Versions ⭐ PRIORITY 2**

**Request:** "Yes, please provide Django ORM versions for all database operations."

**Why Important:**

* Your existing Mingus backend uses Python/Django
* ORM provides better error handling and query optimization
* Easier testing and maintenance
* Consistent with existing codebase patterns

**Specific Request:**

Please convert all SQL operations to Django ORM including:

1. Model Definitions:

- UserFinancialGoal model with proper relationships

- GoalProgressTracking model

- GoalHealthCorrelation model

- Custom model methods for calculations

2. Manager Classes:

- GoalManager with custom querysets

- ProgressManager for analytics

- Methods for complex filtering and aggregation

3. Django Service Classes:

- GoalService class for business logic

- ProgressTrackingService class

- GoalAnalyticsService class

4. Integration with Existing Models:

- Relationships to existing User model

- Integration with health\_checkins table

- Integration with daily\_cashflow table

**3. Example Request/Response Payloads ⭐ PRIORITY 3**

**Request:** "Yes, please provide detailed example request/response payloads for all endpoints."

**Why Essential:**

* Enables frontend development to start immediately
* Prevents API contract changes during frontend development
* Provides clear testing criteria
* Documents expected data structures

**Specific Request:**

Please provide complete request/response examples for:

1. POST /api/goals/setup (Onboarding goal creation)

- Request payload with multiple goals

- Success response with validation results

- Error responses for various failure modes

2. GET /api/goals/dashboard (Dashboard data)

- Response with goal summaries, progress, analytics

- Include goal status indicators

- Include next milestone information

3. POST /api/goals/<id>/contribute (Progress tracking)

- Request payload for manual contribution

- Request payload for automatic detection

- Response with updated progress and analysis

4. GET /api/goals/progress-analysis (Health correlations)

- Response with goal-health correlation data

- Include trend analysis and insights

- Include recommendations based on patterns

5. Error Response Standards:

- Validation error format

- Business logic error format

- Database error handling

- Consistent error code structure

**4. Additional Implementation Requests**

**A. Integration Preparation**

**Request:** "Please also provide the foundation for cash flow integration."

Specifically:

1. Database views or queries for cash flow impact analysis

2. Service methods that will be called by daily\_cashflow calculations

3. Data structures for goal contribution scheduling

4. Placeholder methods for transaction-to-goal mapping

**B. Testing Framework**

**Request:** "Please include comprehensive test cases for all service layer logic."

Including:

1. Unit tests for all calculation methods

2. Integration tests for database operations

3. Test data fixtures for various goal scenarios

4. Performance benchmarks for complex queries

**C. Performance Optimization**

**Request:** "Please include database indexing strategy and query optimization."

Specifically:

1. Recommended database indexes for performance

2. Query optimization for dashboard data loading

3. Caching strategy for frequently accessed goal data

4. Pagination strategy for goal lists and progress history

**5. Immediate Next Steps Priority Order**

**Step 1: Apply Migration + Full DB Logic (Today)**

* Execute the Supabase migration
* Implement complete service-layer DB logic
* Test all CRUD operations with sample data

**Step 2: Django ORM Implementation (Tomorrow)**

* Convert to Django ORM models and services
* Integrate with existing Mingus backend structure
* Test ORM performance vs raw SQL

**Step 3: API Contract Definition (Day 3)**

* Implement API endpoints using service layer
* Validate request/response payloads
* Create API documentation

**Step 4: Integration Foundation (Day 4)**

* Implement cash flow integration hooks
* Prepare health correlation data structures
* Set up special events integration points

**6. Code Review Checkpoints**

Before proceeding to frontend development, ensure:

* [ ] All database operations handle edge cases
* [ ] Service layer methods include proper error handling
* [ ] API responses are consistent and well-documented
* [ ] Integration points are clearly defined
* [ ] Performance benchmarks are established
* [ ] Test coverage is comprehensive

**7. Risk Mitigation**

**If Implementation Takes Longer Than Expected:**

1. **Priority Focus:** Complete CRUD operations first
2. **Defer Complex Analytics:** Start with basic goal tracking
3. **Simplify Integration:** Begin with manual progress tracking
4. **Parallel Frontend:** Use mock data to start UI development

**Quality Gates:**

* Database operations must handle 1000+ goals per user
* API responses must be under 200ms
* All business logic must have 90%+ test coverage
* Integration points must be clearly documented

**Sample Response Template**

**Recommended Response to Developer:**

Thanks for the implementation! Please provide:

1. ✅ YES - Full service-layer DB logic implementation with all CRUD operations and business logic methods

2. ✅ YES - Django ORM versions for consistency with existing Mingus backend

3. ✅ YES - Complete example request/response payloads for all endpoints

4. ✅ ADDITIONAL - Foundation for cash flow integration including service methods and data structures

5. ✅ ADDITIONAL - Comprehensive test cases and performance optimization recommendations

This will enable us to proceed directly to Phase 3 (Integration Layer) without any backend rework.