Weekly Check-in System Implementation Prompts for Cursor

Updated with Integrated Spending Estimates

How to Use These Prompts

Each prompt is designed to be copy-pasted directly into Cursor. They are sequenced in implementation order with dependencies noted. After each prompt, review the generated code, test it, and commit before moving to the next prompt.

Key Change: This version integrates spending estimates directly into the Weekly Check-in, eliminating the need for separate expense tracking or bank integration for MVP.

PHASE 1: DATABASE & BACKEND FOUNDATION

Prompt 1.1: Database Schema Creation

```
Create the database migration and SQLAlchemy models for the Weekly Check-in System with integrated spending estimates.

CONTEXT:
- This is a Flask application using SQLAlchemy with PostgreSQL
- Existing models are in backend/models/
- Migrations use Alembic in backend/migrations/
- The User model already exists with id as UUID primary key
- This system collects BOTH wellness data AND spending estimates in a single weekly check-in

CREATE THESE TABLES:

1. `weekly_checkins` - Core check-in data with wellness AND spending
   - id: UUID primary key
   - user_id: UUID foreign key to users
   - week_ending_date: DATE (always a Sunday, unique with user_id)

   -- PHYSICAL WELLNESS
   - exercise_days: INTEGER (0-7)
   - exercise_intensity: VARCHAR(10) enum ['light', 'moderate', 'intense', null]
   - sleep_quality: INTEGER (1-10)

   -- MENTAL WELLNESS
```

- meditation_minutes: INTEGER (0-999)
- stress_level: INTEGER (1-10)
- overall_mood: INTEGER (1-10)

-- RELATIONAL WELLNESS
- relationship_satisfaction: INTEGER (1-10)
- social_interactions: INTEGER (0+)

-- FINANCIAL FEELINGS
- financial_stress: INTEGER (1-10)
- spending_control: INTEGER (1-10)

-- WEEKLY SPENDING ESTIMATES (NEW)
- groceries_estimate: DECIMAL(10,2) nullable
- dining_estimate: DECIMAL(10,2) nullable
- entertainment_estimate: DECIMAL(10,2) nullable
- shopping_estimate: DECIMAL(10,2) nullable
- transport_estimate: DECIMAL(10,2) nullable
- utilities_estimate: DECIMAL(10,2) nullable
- other_estimate: DECIMAL(10,2) nullable

-- TAGGED SPENDING (NEW - direct questions about problematic spending)
- impulse_spending: DECIMAL(10,2) nullable (amount spent impulsively)
- stress_spending: DECIMAL(10,2) nullable (amount spent due to stress)
- celebration_spending: DECIMAL(10,2) nullable (amount spent celebrating)
- had_impulse_purchases: BOOLEAN default false
- had_stress_purchases: BOOLEAN default false
- biggest_unnecessary_purchase: DECIMAL(10,2) nullable
- biggest_unnecessary_category: VARCHAR(50) nullable

-- REFLECTION
- wins: TEXT (optional)
- challenges: TEXT (optional)

-- METADATA
- completed_at: TIMESTAMP
- completion_time_seconds: INTEGER
- reminder_count: INTEGER default 0

2. `recurring_expenses` - User's fixed monthly expenses (set up once)
- id: UUID primary key
- user_id: UUID foreign key
- name: VARCHAR(100) (e.g., "Rent", "Car Payment")
- amount: DECIMAL(10,2)

- category: VARCHAR(50) enum ['housing', 'transportation', 'insurance', 'debt', 'subscription', 'utilities', 'other']
- due_day: INTEGER (1-31, day of month)
- frequency: VARCHAR(20) enum ['monthly', 'biweekly', 'weekly', 'quarterly', 'annual']
- is_active: BOOLEAN default true
- created_at: TIMESTAMP
- updated_at: TIMESTAMP

3. `user_income` - User's income sources
- id: UUID primary key
- user_id: UUID foreign key
- source_name: VARCHAR(100) (e.g., "Primary Job", "Side Hustle")
- amount: DECIMAL(10,2)
- frequency: VARCHAR(20) enum ['monthly', 'biweekly', 'weekly', 'annual']
- pay_day: INTEGER nullable (day of month for monthly, or 1/15 for biweekly)
- is_active: BOOLEAN default true
- created_at: TIMESTAMP

4. `wellness_scores` - Computed weekly scores
- id: UUID primary key
- user_id: UUID foreign key
- week_ending_date: DATE
- checkin_id: UUID foreign key to weekly_checkins
- physical_score: DECIMAL(5,2)
- mental_score: DECIMAL(5,2)
- relational_score: DECIMAL(5,2)
- financial_feeling_score: DECIMAL(5,2)
- overall_wellness_score: DECIMAL(5,2)
- physical_change: DECIMAL(5,2)
- mental_change: DECIMAL(5,2)
- relational_change: DECIMAL(5,2)
- overall_change: DECIMAL(5,2)
- calculated_at: TIMESTAMP

5. `wellness_finance_correlations` - Correlation analysis results
- id: UUID primary key
- user_id: UUID foreign key
- start_date: DATE
- end_date: DATE
- weeks_analyzed: INTEGER
- stress_vs_impulse_spending: DECIMAL(4,3)
- stress_vs_total_spending: DECIMAL(4,3)
- exercise_vs_spending_control: DECIMAL(4,3)
- sleep_vs_dining_spending: DECIMAL(4,3)
- mood_vs_entertainment_spending: DECIMAL(4,3)

- mood_vs_shopping_spending: DECIMAL(4,3)
- meditation_vs_impulse_spending: DECIMAL(4,3)
- relationship_vs_discretionary_spending: DECIMAL(4,3)
- data_points: INTEGER
- confidence_level: VARCHAR(10) enum ['low', 'medium', 'high']
- strongest_correlation_type: VARCHAR(50)
- strongest_correlation_value: DECIMAL(4,3)
- calculated_at: TIMESTAMP

6. `user_streaks` - Track check-in consistency
- id: UUID primary key
- user_id: UUID foreign key (unique)
- current_streak: INTEGER default 0
- longest_streak: INTEGER default 0
- last_checkin_date: DATE
- total_checkins: INTEGER default 0
- updated_at: TIMESTAMP

7. `user_spending_baselines` - Computed averages for "more/less than usual" comparisons
- id: UUID primary key
- user_id: UUID foreign key (unique)
- avg_groceries: DECIMAL(10,2)
- avg_dining: DECIMAL(10,2)
- avg_entertainment: DECIMAL(10,2)
- avg_shopping: DECIMAL(10,2)
- avg_transport: DECIMAL(10,2)
- avg_total_variable: DECIMAL(10,2)
- avg_impulse: DECIMAL(10,2)
- weeks_of_data: INTEGER
- last_calculated: TIMESTAMP

Create:
1. SQLAlchemy models in backend/models/wellness.py
2. SQLAlchemy models in backend/models/financial_setup.py (for recurring_expenses, user_income)
3. Alembic migration file
4. Add proper indexes for user_id and date columns
5. Add CHECK constraints for value ranges
6. Add relationships to User model

Prompt 1.2: Wellness Score Calculator Service

Create the Wellness Score Calculator service that computes wellness scores from check-in data.

CONTEXT:
- Flask application with services in backend/services/
- This service should be stateless and testable
- All calculations use only user-provided data (no external APIs)

CREATE: backend/services/wellness_score_service.py

REQUIREMENTS:

1. WellnessScoreCalculator class with these methods:

   calculate_physical_score(checkin: dict) -> float
   - Exercise days (0-7) → 0-40 points: (exercise_days / 7) * 40
   - Exercise intensity → 0-20 points: light=10, moderate=15, intense=20 (only if exercise_days > 0)
   - Sleep quality (1-10) → 0-40 points: ((sleep_quality - 1) / 9) * 40
   - Returns 0-100 score

   calculate_mental_score(checkin: dict) -> float
   - Meditation minutes → 0-30 points: min(meditation_minutes, 60) / 60 * 30 (caps at 60 min)
   - Stress level (inverted, 1=low stress, 10=high stress) → 0-35 points: ((11 - stress_level - 1) / 9) * 35
   - Overall mood → 0-35 points: ((overall_mood - 1) / 9) * 35
   - Returns 0-100 score

   calculate_relational_score(checkin: dict) -> float
   - Relationship satisfaction → 0-60 points: ((relationship_satisfaction - 1) / 9) * 60
   - Social interactions → 0-40 points: min(social_interactions, 10) / 10 * 40 (caps at 10)
   - Returns 0-100 score

   calculate_financial_feeling_score(checkin: dict) -> float
   - Financial stress (inverted) → 0-50 points: ((11 - financial_stress - 1) / 9) * 50
   - Spending control → 0-50 points: ((spending_control - 1) / 9) * 50
   - Returns 0-100 score

   calculate_overall_wellness(checkin: dict, weights: dict = None) -> dict
   - Default weights: physical=0.30, mental=0.30, relational=0.20, financial_feeling=0.20
   - Returns dict with all component scores and overall_wellness_score
   - All scores rounded to 2 decimal places

   calculate_week_over_week_changes(current_scores: dict, previous_scores: dict) -> dict

- Calculate change for each component and overall
- Returns dict with physical_change, mental_change, relational_change, overall_change

2. Helper method get_week_ending_date(date) to get the Sunday for any given date

3. Input validation that raises ValueError for out-of-range values

4. Unit tests in tests/test_wellness_score_service.py covering:
- Edge cases (all zeros, all max values)
- Normal calculations
- Week-over-week changes
- Invalid input handling

Include docstrings explaining each calculation formula.

Prompt 1.3: Correlation Engine Service (Updated for Spending Estimates)

Create the Wellness-Finance Correlation Engine that finds patterns between wellness and spending estimates.

CONTEXT:
- Flask application with services in backend/services/
- Uses spending estimates collected IN the weekly check-in (not separate expense tracking)
- Needs minimum 4 weeks of data to calculate correlations
- Must handle cases where user has insufficient data gracefully

CREATE: backend/services/correlation_engine_service.py

REQUIREMENTS:

1. CorrelationResult dataclass:
- correlation: float (-1 to +1)
- strength: str ('weak', 'moderate', 'strong')
- direction: str ('positive', 'negative', 'none')
- data_points: int
- confidence: str ('low', 'medium', 'high')
- insight: str (human-readable insight message)
- dollar_impact: float (estimated $ difference between good/bad weeks)

2. WellnessFinanceCorrelator class:

Constants:

- MIN_DATA_POINTS = 4
- CONFIDENCE_THRESHOLDS = {'low': 4, 'medium': 8, 'high': 12}

pearson_correlation(x: List[float], y: List[float]) -> Optional[float]
- Calculate Pearson correlation coefficient
- Return None if insufficient data or zero variance
- Handle edge cases (all same values, mismatched lengths)

interpret_correlation(r: float, n: int) -> CorrelationResult
- strength: |r| < 0.3 = 'weak', 0.3-0.6 = 'moderate', >0.6 = 'strong'
- direction: |r| < 0.1 = 'none', r > 0 = 'positive', r < 0 = 'negative'
- confidence based on data points per CONFIDENCE_THRESHOLDS

CORRELATIONS USING CHECK-IN SPENDING DATA:

correlate_stress_and_impulse(checkins: List[dict]) -> CorrelationResult
- Correlate stress_level with impulse_spending amount from same check-in
- Generate insight like "High stress weeks show $X more impulse spending"

correlate_stress_and_total_spending(checkins: List[dict]) -> CorrelationResult
- Correlate stress_level with total variable spending (sum of all estimates)
- Calculate total as: groceries + dining + entertainment + shopping + transport + other

correlate_exercise_and_spending_control(checkins: List[dict]) -> CorrelationResult
- Correlate exercise_days with spending_control feeling (1-10)
- Generate insight about exercise and financial discipline

correlate_sleep_and_dining(checkins: List[dict]) -> CorrelationResult
- Correlate sleep_quality (inverted) with dining_estimate
- Test hypothesis: poor sleep → more takeout/delivery

correlate_mood_and_entertainment(checkins: List[dict]) -> CorrelationResult
- Correlate overall_mood with entertainment_estimate
- Detect both "happy spending" and "sad retail therapy" patterns

correlate_mood_and_shopping(checkins: List[dict]) -> CorrelationResult
- Correlate overall_mood (and inverted) with shopping_estimate
- Emotional shopping detection

correlate_meditation_and_impulse(checkins: List[dict]) -> CorrelationResult
- Correlate meditation_minutes with impulse_spending
- Test if mindfulness reduces impulse purchases

correlate_relationship_and_discretionary(checkins: List[dict]) -> CorrelationResult

- Correlate relationship_satisfaction with discretionary spending (entertainment + shopping)
- Test relationship stress → retail therapy hypothesis

generate_all_correlations(user_id: str, checkins: List[dict]) -> Dict[str, CorrelationResult]
- Run all correlation analyses using check-in data
- Return dict keyed by correlation type
- Only include correlations where sufficient non-null spending data exists

get_strongest_correlation(correlations: Dict) -> Tuple[str, CorrelationResult]
- Find the correlation with highest absolute value that has medium+ confidence

3. Helper methods:

calculate_total_variable_spending(checkin: dict) -> float
- Sum all spending estimate fields, treating None as 0

calculate_discretionary_spending(checkin: dict) -> float
- Sum entertainment + shopping estimates

_calculate_high_low_difference(checkins: List[dict], wellness_field: str, spending_field: str, threshold: float) -> float
- Calculate average spending difference between high and low wellness values
- Used for generating dollar-amount insights
- Example: average impulse spending when stress >= 7 vs stress <= 4

4. All insights should be:
- Supportive, not judgmental
- Actionable when possible
- Include specific dollar amounts from the user's data
- Example: "Your high-stress weeks (7+) average $85 more in impulse spending than calm weeks."

5. Unit tests in tests/test_correlation_engine_service.py

Prompt 1.4: Insight Generator Service

Create the Insight Generator that produces actionable wellness-finance insights from check-in data.

CONTEXT:
- Flask application with services in backend/services/
- Uses WellnessScoreCalculator and WellnessFinanceCorrelator
- Generates personalized insights based on current check-in and historical patterns
- All spending data comes from weekly check-in estimates (not external bank data)

- Target audience: African American professionals ages 25-35 (per Mingus Master Prompt)

CREATE: backend/services/insight_generator_service.py

REQUIREMENTS:

1. InsightType enum: CORRELATION, TREND, ANOMALY, ACHIEVEMENT, RECOMMENDATION, SPENDING_PATTERN

2. WellnessInsight dataclass:
   - type: InsightType
   - title: str (short, engaging, with emoji)
   - message: str (2-3 sentences max)
   - data_backing: str (the numbers behind the insight)
   - action: str (specific suggested action)
   - priority: int (1-5, 1 being highest)
   - category: str ('physical', 'mental', 'relational', 'financial', 'spending')
   - dollar_amount: float (optional, the $ impact if applicable)

3. InsightGenerator class:

   generate_weekly_insights(
      current_checkin: dict,
      previous_checkins: List[dict],
      correlations: Dict[str, CorrelationResult],
      spending_baselines: dict
) -> List[WellnessInsight]
   - Generate prioritized list of insights
   - Return top 5 insights sorted by priority
   - Call these internal methods:

   _correlation_insights(correlations) -> List[WellnessInsight]
   - Only include correlations with medium+ confidence and moderate+ strength
   - Priority 2 for strong, 3 for moderate
   - Include dollar amounts when available

   _trend_insights(current_checkin, previous_checkins) -> List[WellnessInsight]
   - Detect 3+ week trends (improving or declining) in:
     - exercise_days, stress_level, meditation_minutes, overall_mood
     - impulse_spending, total spending
   - Priority 2 for positive trends, 3 for concerning trends

   _spending_pattern_insights(current_checkin, previous_checkins, baselines) -> List[WellnessInsight]
   - Compare current week spending to user's own averages

- Flag if spending is significantly above/below normal
- Highlight if impulse or stress spending is elevated
- Example: "Your shopping this week ($180) is 2x your usual average"

_anomaly_insights(current_checkin) -> List[WellnessInsight]
- Flag unusual patterns in the CURRENT check-in:
  - High stress (≥7) + reported impulse spending
  - Low mood + high shopping/entertainment
  - Poor sleep + high dining out
- Priority 1 for concerning anomalies (real-time alerts)

_achievement_insights(current_checkin, previous_checkins) -> List[WellnessInsight]
- Celebrate milestones:
  - First time reporting $0 impulse spending
  - 3 weeks in a row with no stress spending
  - Lowest total spending week
  - 5+ exercise days
- Priority 1-2 for achievements

_recommendation_insights(current_checkin, correlations) -> List[WellnessInsight]
- If user has strong stress-impulse correlation AND current stress ≥ 6:
  "Your pattern shows stress leads to impulse buys. Consider a 24-hour pause this week."
- If current spending is trending up AND mood is trending down:
  "Spending is up while mood is down. Check in with yourself before the next purchase."
- Priority 1 for actionable recommendations based on current state

4. Spending baseline comparison helper:

compare_to_baseline(current_value: float, baseline: float) -> str
- Returns: 'much_lower' (<50%), 'lower' (50-80%), 'normal' (80-120%), 'higher' (120-150%), 'much_higher' (>150%)

5. Tone guidelines (implement in message generation):
- Supportive, never judgmental ("Notice" not "You failed")
- Celebrate progress, no matter how small
- Use "your data shows" to ground insights in facts
- Suggest actions, don't demand them
- Use appropriate emojis sparingly
- Include specific dollar amounts to make insights tangible

6. Example insights to generate:

CORRELATION:
- "🔗 Stress & Spending Link" - "Your high-stress weeks average $85 more in impulse spending. This week's stress: 7/10."

SPENDING_PATTERN:
- "📊 Shopping Spike" - "Your shopping this week ($180) is 2.3x your usual $78 average."
- "💚 Spending Win" - "Your total spending this week is 15% below your average. Nice control!"

ANOMALY:
- "⚡ Stress Spending Alert" - "You reported high stress (8/10) and $120 in impulse purchases. These often go together for you."

TREND:
- "📈 Impulse Trend" - "Your impulse spending has increased for 3 weeks: $40 → $65 → $95."
- "📉 Great Progress" - "Your stress spending has been $0 for 4 weeks straight!"

ACHIEVEMENT:
- "🏆 Mindful Month" - "4 weeks of meditation + your lowest impulse spending month!"

RECOMMENDATION:
- "🛡️ Spending Shield" - "Stress is 8/10 today. Based on your pattern, consider waiting 24 hours on non-essentials."

7. Unit tests in tests/test_insight_generator_service.py

Prompt 1.5: Weekly Check-in API Endpoints (Updated with Spending)

Create the REST API endpoints for the Weekly Check-in System with integrated spending estimates.

CONTEXT:
- Flask application with API routes in backend/api/
- Uses Flask-RESTful or Flask blueprints
- JWT authentication is already implemented (use @jwt_required decorator)
- Current user available via get_jwt_identity() or current_user
- Check-in includes BOTH wellness metrics AND spending estimates

CREATE: backend/api/wellness_checkin_api.py

ENDPOINTS:

1. POST /api/wellness/checkin
   - Submit a weekly check-in with wellness AND spending data
   - Request body: {
     // Wellness (required)
     exercise_days: int (0-7),

exercise_intensity: str or null,
sleep_quality: int (1-10),
meditation_minutes: int (0+),
stress_level: int (1-10),
overall_mood: int (1-10),
relationship_satisfaction: int (1-10),
social_interactions: int (0+),
financial_stress: int (1-10),
spending_control: int (1-10),

// Spending estimates (all optional, null = didn't track)
groceries_estimate: decimal or null,
dining_estimate: decimal or null,
entertainment_estimate: decimal or null,
shopping_estimate: decimal or null,
transport_estimate: decimal or null,
utilities_estimate: decimal or null,
other_estimate: decimal or null,

// Tagged spending (optional)
had_impulse_purchases: bool,
impulse_spending: decimal or null,
had_stress_purchases: bool,
stress_spending: decimal or null,
celebration_spending: decimal or null,
biggest_unnecessary_purchase: decimal or null,
biggest_unnecessary_category: str or null,

// Reflection (optional)
wins: str or null,
challenges: str or null,

// Metadata
completion_time_seconds: int
}
- Validates all fields are in correct ranges
- Calculates week_ending_date (current week's Sunday)
- Prevents duplicate check-ins for same week (return 409 Conflict)
- Triggers: wellness score calculation, correlation update, baseline update
- Updates user streak
- Returns: { checkin_id, wellness_scores, streak_info, insights }

2. GET /api/wellness/checkin/current-week
- Get current week's check-in if exists

- Returns check-in data (wellness + spending) or 404 if not completed

3. GET /api/wellness/checkin/history?weeks=12
   - Get check-in history for specified number of weeks
   - Default 12 weeks, max 52 weeks
   - Returns array of check-ins with wellness_scores
   - Includes spending data for each week

4. GET /api/wellness/scores/latest
   - Get most recent wellness scores
   - Returns scores with week-over-week changes

5. GET /api/wellness/scores/history?weeks=12
   - Get wellness score history for charts
   - Returns array of { week_ending_date, physical_score, mental_score, relational_score, financial_feeling_score, overall_wellness_score }

6. GET /api/wellness/spending/history?weeks=12
   - Get spending estimate history for charts
   - Returns array of { week_ending_date, groceries, dining, entertainment, shopping, transport, total, impulse, stress }

7. GET /api/wellness/spending/baselines
   - Get user's spending baselines (averages)
   - Returns: { avg_groceries, avg_dining, avg_entertainment, avg_shopping, avg_transport, avg_total, avg_impulse, weeks_of_data }

8. GET /api/wellness/insights
   - Get current insights based on latest check-in and correlations
   - Returns array of WellnessInsight objects
   - Returns encouragement message if insufficient data (< 4 weeks)

9. GET /api/wellness/correlations
   - Get all computed correlations for user
   - Returns correlation data with confidence levels and dollar impacts
   - Returns message if insufficient data

10. POST /api/wellness/correlations/refresh
    - Manually trigger correlation recalculation
    - Also recalculates spending baselines
    - Returns updated correlations

11. GET /api/wellness/streak
    - Get user's streak information
    - Returns { current_streak, longest_streak, last_checkin_date, total_checkins }

12. GET /api/wellness/summary
    - Get a dashboard summary combining multiple data points
    - Returns: {
        current_scores: {...},
        streak: {...},
        spending_this_week: {...},
        spending_vs_baseline: {...},
        top_insights: [...],
        weeks_of_data: int
      }

ALSO CREATE:
- Request validation schemas using marshmallow or pydantic
- Proper error handling with meaningful messages
- Rate limiting on POST endpoints
- Register blueprint in backend/api/__init__.py

Include OpenAPI/Swagger documentation comments for each endpoint.

Prompt 1.6: Financial Setup API (Recurring Expenses & Income)

Create the API endpoints for one-time financial setup (recurring expenses and income).

CONTEXT:
- Flask application with existing user system
- Users set this up once during onboarding, can edit later
- This provides the "fixed" portion of their monthly budget
- Variable spending comes from weekly check-in estimates

CREATE: backend/api/financial_setup_api.py

ENDPOINTS:

1. GET /api/financial/setup/status
    - Check if user has completed financial setup
    - Returns: { is_complete: bool, has_income: bool, has_expenses: bool, monthly_income: decimal, monthly_fixed_expenses: decimal }

2. POST /api/financial/income
    - Add an income source

- Request body: {
  source_name: str (e.g., "Primary Job"),
  amount: decimal,
  frequency: str enum ['monthly', 'biweekly', 'weekly', 'annual'],
  pay_day: int or null
 }
- Returns created income record

3. GET /api/financial/income
- Get all active income sources for user
- Returns array of income records with calculated monthly_equivalent

4. PUT /api/financial/income/{id}
- Update an income source
- Returns updated record

5. DELETE /api/financial/income/{id}
- Soft delete (set is_active = false)

6. POST /api/financial/expenses/recurring
- Add a recurring expense
- Request body: {
  name: str (e.g., "Rent"),
  amount: decimal,
  category: str enum ['housing', 'transportation', 'insurance', 'debt', 'subscription', 'utilities', 'childcare', 'other'],
  due_day: int (1-31),
  frequency: str enum ['monthly', 'biweekly', 'weekly', 'quarterly', 'annual']
 }
- Returns created expense record

7. GET /api/financial/expenses/recurring
- Get all active recurring expenses for user
- Returns array with calculated monthly_equivalent for each
- Include total monthly fixed expenses

8. PUT /api/financial/expenses/recurring/{id}
- Update a recurring expense

9. DELETE /api/financial/expenses/recurring/{id}
- Soft delete (set is_active = false)

10. GET /api/financial/summary
- Get complete financial picture
- Returns: {

monthly_income: decimal,
monthly_fixed_expenses: decimal,
monthly_discretionary: decimal (income - fixed),
income_sources: [...],
recurring_expenses: [...],
expenses_by_category: { housing: X, transportation: Y, ... }
}

11. POST /api/financial/setup/complete
- Mark financial setup as complete (for onboarding flow)
- Validates user has at least one income source
- Returns setup status

HELPER METHODS:
- calculate_monthly_equivalent(amount, frequency) -> decimal
- Convert any frequency to monthly amount
- weekly * 4.33, biweekly * 2.17, quarterly / 3, annual / 12

Register in backend/api/__init__.py

Prompt 1.7: Spending Baseline Calculator Service

Create a service to calculate and update user spending baselines for comparison.

CONTEXT:
- Users report weekly spending estimates in their check-ins
- We need to calculate their "normal" spending to detect anomalies
- Baselines should update after each check-in
- Used for insights like "Your shopping this week is 2x your average"

CREATE: backend/services/spending_baseline_service.py

REQUIREMENTS:

1. SpendingBaselineService class:

calculate_baselines(user_id: str, checkins: List[dict]) -> dict
- Calculate average spending for each category across all check-ins
- Only include check-ins where the category was reported (not null)
- Return: {
avg_groceries: float,

avg_dining: float,
avg_entertainment: float,
avg_shopping: float,
avg_transport: float,
avg_utilities: float,
avg_other: float,
avg_total_variable: float,
avg_impulse: float,
avg_stress: float,
weeks_of_data: int
}

update_baselines(user_id: str) -> dict
- Fetch all check-ins for user
- Recalculate baselines
- Save to user_spending_baselines table
- Return updated baselines

get_baselines(user_id: str) -> dict
- Get current baselines from database
- Return None if no baselines exist yet

compare_to_baseline(user_id: str, current_checkin: dict) -> dict
- Compare current week's spending to baselines
- Return: {
groceries: { current: 120, baseline: 100, difference: 20, percent_change: 20, status: 'higher' },
dining: { current: 80, baseline: 60, difference: 20, percent_change: 33, status: 'much_higher' },
...
total: { current: 450, baseline: 380, difference: 70, percent_change: 18, status: 'higher' },
impulse: { current: 50, baseline: 30, difference: 20, percent_change: 67, status: 'much_higher' }
}

Status values: 'much_lower' (<50%), 'lower' (50-80%), 'normal' (80-120%), 'higher' (120-150%), 'much_higher' (>150%)

2. Helper methods:

_safe_average(values: List[float]) -> Optional[float]
- Calculate average, excluding None values
- Return None if no valid values

_calculate_percent_change(current: float, baseline: float) -> float
- Handle zero baseline edge case
- Return percentage change

_get_status(percent_change: float) -> str
- Categorize the change into status buckets

3. Auto-update trigger:
- Call update_baselines() after each check-in submission
- Consider using rolling average (last 8-12 weeks) instead of all-time average
- This prevents old data from skewing baselines

4. Minimum data requirement:
- Need at least 3 weeks of data for meaningful baselines
- Return special status for insufficient data

Include unit tests in tests/test_spending_baseline_service.py

---

PHASE 2: FRONTEND IMPLEMENTATION

Prompt 2.1: Weekly Check-in Form Component (Updated with Spending)

Create the Weekly Check-in form component with a 6-step mobile-first flow including spending estimates.

CONTEXT:
- React 18 with TypeScript
- Tailwind CSS for styling
- Components in frontend/src/components/
- Use existing design system colors (violet primary, dark background)
- Mobile-first responsive design
- This form collects BOTH wellness data AND spending estimates

CREATE: frontend/src/components/wellness/WeeklyCheckinForm.tsx

REQUIREMENTS:

1. Multi-step form with progress indicator (Step 1 of 6)

2. Step 1: Physical Wellness (60 seconds)
- "How many days did you exercise this week?" - Tap buttons 0-7
- "How intense were your workouts?" - Three option cards: Light 🚶 , Moderate 🏃 , Intense 💪
- "How was your sleep quality?" - Slider 1-10 with emoji endpoints 😫 to 😴
- Skip intensity question if exercise_days = 0

3. Step 2: Mental Wellness (60 seconds)

- "Minutes of meditation/prayer/mindfulness?" - Preset buttons: 0, 15, 30, 45, 60, 90, 120+
- "What was your average stress level?" - Slider 1-10, 😌 to 😰
- "Overall mood this week?" - 5 emoji buttons: 😢 😕 😐 🙂 😊 (maps to 2,4,6,8,10)

4. Step 3: Relationships (45 seconds)
- "How satisfied are you with your key relationships?" - Slider 1-10, 💔 to 💖
- "Meaningful social interactions this week?" - Preset buttons: 0, 1-2, 3-5, 6-10, 10+

5. Step 4: Financial Feelings (45 seconds)
- "How stressed about money were you this week?" - Slider 1-10, 😌 to 😰
- "How in control of spending did you feel?" - Slider 1-10, 😬 to 💪

6. Step 5: Weekly Spending (NEW - 90 seconds)
Header: "Let's estimate your spending this week 💸 "
Subtitle: "Rough estimates are fine! This helps us find patterns."

Category inputs with suggested ranges (show user's baseline if available):
- "Groceries" - Number input with placeholder "~$100" or "Your avg: $95"
- "Dining & Takeout" - Number input
- "Entertainment" - Number input
- "Shopping" - Number input
- "Gas & Transport" - Number input
- "Other" - Number input (optional)

Quick estimate buttons for each: [Skip] [~$25] [~$50] [~$100] [~$150] [~$200+]

Separator line, then:

"Any impulse purchases this week?"
- Toggle: [No] [Yes, about $___]
- If yes, show amount input

"Any stress-related spending?"
- Toggle: [No] [Yes, about $___]
- If yes, show amount input

Optional: "Biggest purchase you regret?" - Amount + category dropdown

7. Step 6: Reflection (optional, 60 seconds)
- "One win from this week?" - Text input, 200 char max
- "One challenge you faced?" - Text input, 200 char max
- [Skip & Finish] button prominent

8. UI/UX Requirements:

- Large touch targets (min 44px)
- Smooth transitions between steps
- Back button on steps 2-6
- Progress bar showing completion
- Haptic feedback on mobile (if available)
- Form state persisted to localStorage (in case of accidental close)
- Timer tracking completion_time_seconds
- Confirmation screen after submission showing:
  - Wellness scores
  - Total spending this week
  - Comparison to their average (if available)
  - Any immediate insights

9. Spending input UX details:
   - Show user's baseline average as placeholder/hint if available
   - "About average", "More than usual", "Less than usual" quick buttons as alternative to exact amounts
   - Convert relative inputs to dollar estimates based on baseline
   - Allow $0 as valid input (different from skipping)
   - Null/skip means "I don't know" - won't be included in averages

10. State management:
    - Use React Hook Form or useState
    - Validate each step before allowing next
    - Track start time for completion_time_seconds
    - Handle partial spending data (some categories filled, others skipped)

11. API integration:
    - POST to /api/wellness/checkin on submit
    - Handle loading and error states
    - Show success animation on completion
    - Navigate to dashboard or show results modal

12. Accessibility:
    - Proper ARIA labels
    - Keyboard navigation
    - Screen reader friendly
    - Currency inputs formatted properly

Create supporting components:
- StepIndicator.tsx - Progress dots/bar
- SliderInput.tsx - Reusable 1-10 slider with emoji endpoints
- EmojiSelector.tsx - Row of emoji buttons
- NumberSelector.tsx - Row of number/range buttons
- SpendingInput.tsx - Currency input with quick-select buttons

- SpendingCategory.tsx - Single spending category row
- ToggleWithAmount.tsx - Yes/No toggle that reveals amount input

---

Prompt 2.2: Wellness Score Card Component

Create the Wellness Score Card component for the dashboard.

CONTEXT:
- React 18 with TypeScript
- Tailwind CSS styling
- This is a key dashboard component showing the user's wellness status
- Should be visually engaging and easy to understand at a glance

CREATE: frontend/src/components/wellness/WellnessScoreCard.tsx

REQUIREMENTS:

1. Props interface:

```
interface WellnessScores {
  physical_score: number;
  mental_score: number;
  relational_score: number;
  financial_feeling_score: number;
  overall_wellness_score: number;
  physical_change: number;
  mental_change: number;
  relational_change: number;
  overall_change: number;
  week_ending_date: string;
}
```

2. Layout:
   - Hero section: Large overall score (0-100) with circular progress ring
   - Change indicator: ↑ +5.2 from last week (green), ↓ -3.1 (red), → 0 (gray)
   - Four category cards in 2x2 grid below:
     - Physical 💪 with score and mini change indicator
     - Mental 🧠 with score and mini change indicator
     - Relational ❤️ with score and mini change indicator
     - Financial Feel 💰 with score and mini change indicator

3. Visual design:

- Score color coding:
  - 75-100: Green (#10B981) - "Thriving"
  - 50-74: Yellow (#F59E0B) - "Growing"
  - 25-49: Orange (#F97316) - "Building"
  - 0-24: Red (#EF4444) - "Needs attention"
- Animated number counting on load
- Subtle pulse animation on the overall score ring
- Category cards have colored left border matching their score

4. Empty state:
- If no scores yet, show "Complete your first check-in to see your wellness score"
- CTA button to start check-in

5. Week indicator:
- "Week of Jan 26, 2026" subtitle under the score

6. Responsive:
- Full width on mobile
- Card layout on desktop
- Category grid: 2x2 on mobile, 4x1 row on desktop

Create also:
- CircularProgressRing.tsx - Animated SVG circular progress
- ScoreChangeIndicator.tsx - Arrow with change value
- CategoryScoreCard.tsx - Individual category display

Prompt 2.3: Spending Summary Card Component (NEW)

Create a Spending Summary Card that shows the user's weekly spending at a glance.

CONTEXT:
- React 18 with TypeScript
- Displays spending data from the weekly check-in
- Compares current week to user's baseline averages
- Shows alerts for unusual spending patterns

CREATE: frontend/src/components/wellness/SpendingSummaryCard.tsx

REQUIREMENTS:

1. Props interface:

```
interface SpendingData {
  week_ending_date: string;
  groceries: number | null;
  dining: number | null;
  entertainment: number | null;
  shopping: number | null;
  transport: number | null;
  other: number | null;
  total: number;
  impulse_spending: number | null;
  stress_spending: number | null;
}

interface SpendingBaselines {
  avg_groceries: number;
  avg_dining: number;
  avg_entertainment: number;
  avg_shopping: number;
  avg_transport: number;
  avg_total: number;
  avg_impulse: number;
  weeks_of_data: number;
}

interface Props {
  current: SpendingData;
  baselines: SpendingBaselines | null;
  loading?: boolean;
}
```

2. Layout:
   - Header: "This Week's Spending 💸 " with total amount
   - Comparison badge: "↑ 15% vs your average" or "↓ 8% vs average" or "About average"

   - Category breakdown (horizontal bar or list):
     - Each category shows: name, amount, mini bar, vs-average indicator
     - Highlight categories significantly above average (>150%)
     - Categories with $0 or null shown differently

   - Impulse/Stress spending section (if any):
     - "Impulse: $50" with small warning icon if above average
     - "Stress spending: $30"
     - These are highlighted as "watch" categories

3. Visual design:
   - Clean, not overwhelming
   - Use subtle color coding:
     - Green: below average
     - Gray: about average
     - Yellow: somewhat above average (120-150%)
     - Red: significantly above average (>150%)
   - Small sparkline or mini bar chart for each category

4. Empty/insufficient data states:
   - No spending data: "Complete your check-in to track spending"
   - No baselines yet: "After 3 weeks, we'll show how this compares to your average"

5. Interaction:
   - Tap category to see historical trend (optional, can defer)
   - "See details" link to full spending history view

6. Responsive:
   - Compact on mobile (collapsible category list)
   - Expanded on desktop

Create also:
- SpendingCategoryBar.tsx - Individual category with bar visualization
- SpendingComparisonBadge.tsx - Shows vs-average comparison
- MiniSpendingChart.tsx - Tiny sparkline of recent weeks

Prompt 2.4: Wellness Impact Card Component (Updated)

Create the Wellness Impact Card that displays wellness-finance correlation insights.

CONTEXT:
- React 18 with TypeScript
- This is Mingus's KEY DIFFERENTIATOR - the wellness-finance connection visualization
- Insights now include spending patterns from check-in data
- Must be engaging and make users feel like they're learning about themselves

CREATE: frontend/src/components/wellness/WellnessImpactCard.tsx

REQUIREMENTS:

1. Props interface:

```
interface WellnessInsight {
  type: 'correlation' | 'trend' | 'anomaly' | 'achievement' | 'recommendation' | 'spending_pattern';
  title: string;
  message: string;
  data_backing: string;
  action: string;
  priority: number;
  category: 'physical' | 'mental' | 'relational' | 'financial' | 'spending';
  dollar_amount?: number;
}

interface Props {
  insights: WellnessInsight[];
  loading?: boolean;
  weeksOfData?: number;
}
```

2. Layout:
   - Header: "✨ Your Wellness-Money Connection"
   - List of up to 3 insight cards
   - Each insight card shows:
     - Type icon: 🔗 correlation, 📈 trend, ⚡ anomaly, 🏆 achievement, 💡 recommendation, 📊 spending_pattern
     - Title (bold)
     - Message (2-3 sentences)
     - Dollar amount highlight if applicable (e.g., "$85 more" in larger text)
     - Data backing (smaller, gray, italic)
     - Action box (if action exists)

3. Visual design:
   - Card background colors by category:
     - physical: green-50 with green-200 border
     - mental: purple-50 with purple-200 border
     - relational: pink-50 with pink-200 border
     - financial: blue-50 with blue-200 border
     - spending: amber-50 with amber-200 border
   - Dollar amounts should stand out (larger font, bold)
   - Subtle entrance animation (fade in + slide up)

4. Empty/loading states:
   - Loading: Skeleton cards with shimmer effect
   - No data (< 4 weeks):
     "Complete a few more weekly check-ins to unlock personalized insights!
     We need about 4 weeks of data to find patterns.
     [Progress bar showing X of 4 weeks completed]"

- No insights found:
  "No strong patterns found yet. Keep checking in - insights often emerge after 6-8 weeks!"

5. Priority ordering:
   - Show insights in priority order (1 = highest)
   - Achievement insights get special celebration styling
   - Anomaly/spending alerts shown prominently

6. Accessibility:
   - Screen reader announces new insights
   - Proper heading hierarchy
   - Color not the only indicator of category

Create also:
- InsightCard.tsx - Individual insight display
- InsightProgressBar.tsx - Shows weeks until insights unlock
- InsightSkeleton.tsx - Loading placeholder
- DollarHighlight.tsx - Styled dollar amount display

Prompt 2.5: Check-in Reminder Banner Component

Create a reminder banner component that prompts users to complete their weekly check-in.

CONTEXT:
- React 18 with TypeScript
- Shows on dashboard when check-in is due
- Should be attention-grabbing but not annoying
- Emphasizes both wellness AND spending tracking benefits

CREATE: frontend/src/components/wellness/CheckinReminderBanner.tsx

REQUIREMENTS:

1. Props interface:
```
interface Props {
  lastCheckinDate: string | null;
  currentStreak: number;
  weeksOfData: number;
  onStartCheckin: () => void;
  onDismiss: () => void;
}
```

2. Banner states:

a) Never checked in:
"🌟 Start Your Wellness Journey"
"Take 7 minutes to complete your first check-in. Track your wellness AND spending to discover patterns."
[Start Check-in] button

b) Building data (1-3 check-ins):
"📝 Keep Building Your Profile"
"Check-in #{weeksOfData + 1} - just {4 - weeksOfData} more until we can find your patterns!"
[Continue Building] button

c) Check-in due (4+ check-ins, hasn't done this week):
"📝 Weekly Check-in Time!"
"How was your week? Let's track your wellness and spending."
[Complete Check-in] button

d) Streak at risk (Monday and hasn't checked in):
"🔥 Don't break your {X}-week streak!"
"Complete your check-in before midnight to keep your streak alive!"
[Save My Streak] button (more urgent styling)

e) Already completed this week:
Don't show banner at all

3. Visual design:
- Gradient background (violet to indigo)
- White text
- Large touch-friendly button
- X dismiss button in corner
- Streak fire emoji animation if streak > 2
- Progress indicator showing weeks until insights unlock (if < 4)

4. Dismissal behavior:
- User can dismiss for 24 hours (store in localStorage)
- Reappears if still not completed
- Cannot permanently dismiss if check-in is overdue

5. Animation:
- Slide in from top on mount
- Subtle bounce on the CTA button

Create also:

- StreakCounter.tsx - Animated streak display with fire emoji
- InsightUnlockProgress.tsx - "2 more weeks to unlock insights" indicator

---

Prompt 2.6: Wellness Dashboard Integration (Updated)

Integrate all wellness and spending components into the main dashboard.

CONTEXT:
- Existing dashboard at frontend/src/components/Dashboard.tsx or similar
- Need to add wellness AND spending sections
- Should feel like a unified financial wellness view

MODIFY: frontend/src/components/Dashboard.tsx (or create DashboardWellnessSection.tsx)

REQUIREMENTS:

1. Dashboard layout with wellness + spending integration:

   TOP SECTION (if check-in due):
   - CheckinReminderBanner (full width)

   MAIN GRID:
   Row 1:
   - WellnessScoreCard (left/top)
   - SpendingSummaryCard (right/below)

   Row 2:
   - WellnessImpactCard (full width or left)
   - Quick Stats card (right, optional)

   (existing dashboard content below)

2. Data fetching:
   - Create useWellnessData() hook that fetches:
     - GET /api/wellness/summary (combines multiple endpoints)
     - Current wellness scores
     - Current spending data
     - Spending baselines
     - Insights
     - Streak info
     - Check-in status for current week

- Use React Query or SWR for caching and revalidation
- Refetch after check-in submission

3. State management:
- Track if check-in modal is open
- Track dismissed reminder state
- Optimistic updates after check-in

4. Responsive layout:
Mobile (< 768px):
- Stack everything vertically
- Reminder banner at top
- WellnessScoreCard
- SpendingSummaryCard
- WellnessImpactCard
- Existing dashboard content

Desktop (>= 768px):
- Reminder banner (if shown)
- 2-column grid: Score card left, Spending card right
- Impact card full width below
- Existing dashboard content

5. Loading states:
- Show skeletons for all wellness/spending cards while loading
- Don't block entire dashboard for wellness data

6. Error handling:
- If wellness endpoints fail, show "Unable to load wellness data" with retry
- Don't crash the whole dashboard

7. Check-in flow trigger:
- Clicking "Start Check-in" opens WeeklyCheckinForm in modal or full-screen
- After completion:
  - Refresh all wellness and spending data
  - Show success message with summary
  - Animate score/spending changes
  - Show any new insights that were generated

8. First-time user experience:
- If no check-ins yet, show welcome state
- Explain the wellness-finance connection
- Prominent CTA to complete first check-in
- Preview of what they'll see (blurred/teaser)

CREATE:
- hooks/useWellnessData.ts
- hooks/useSpendingData.ts
- DashboardWellnessSection.tsx (if separating from main dashboard)
- WelcomeState.tsx (for first-time users)

PHASE 3: ONBOARDING & FINANCIAL SETUP

Prompt 3.1: Financial Setup Flow Component

Create the financial setup flow for new user onboarding.

CONTEXT:
- React 18 with TypeScript
- This runs during onboarding, after account creation
- Collects recurring expenses and income (one-time setup)
- Can be skipped and completed later

CREATE: frontend/src/components/onboarding/FinancialSetupFlow.tsx

REQUIREMENTS:

1. Multi-step setup with 3 main sections:

   Step 1: Income Setup
   Header: "Let's start with your income 💵 "

   - Primary income:
     - "What's your take-home pay?" - Amount input
     - "How often are you paid?" - [Weekly] [Bi-weekly] [Monthly]
     - "What day(s)?" - Day selector based on frequency

   - "Add another income source?" link (side hustle, etc.)
   - Shows calculated monthly income at bottom

   Step 2: Fixed Expenses
   Header: "Now your regular bills 📋 "
   Subtitle: "These repeat every month (or so)"

   Quick-add common expenses:

- [+ Rent/Mortgage]
- [+ Car Payment]
- [+ Car Insurance]
- [+ Phone Bill]
- [+ Student Loans]
- [+ Utilities]
- [+ Subscriptions]
- [+ Add Custom]

Each expense form:
- Name (pre-filled for quick-add)
- Amount
- Category (pre-selected for quick-add)
- Due day of month

Running total shown: "Monthly fixed expenses: $1,850"

Step 3: Review & Summary
Header: "Here's your financial picture  "

- Monthly income: $4,200
- Fixed expenses: $1,850 (with breakdown by category)
- Available for variable spending: $2,350

Pie chart or bar showing the breakdown

"This looks right" [Complete Setup] button
"I need to make changes" [Go Back] button

2. Skip option:
- "Skip for now" link on each step
- Explain they can set this up later
- Still allow access to check-ins without this

3. Visual design:
- Clean, focused layout
- One thing at a time
- Progress indicator
- Encouraging copy
- Mobile-optimized inputs (large touch targets, native keyboards)

4. Smart defaults:
- Suggest typical amounts based on selected category
- Remember common subscription amounts ($15 Netflix, etc.)

- Pre-fill category when using quick-add buttons

5. Validation:
- Income must be positive
- Expenses can't exceed income (warning, not blocker)
- At least one income source required to complete

6. API integration:
- POST /api/financial/income for each income source
- POST /api/financial/expenses/recurring for each expense
- POST /api/financial/setup/complete when done

Create supporting components:
- IncomeForm.tsx
- RecurringExpenseForm.tsx
- ExpenseQuickAdd.tsx
- FinancialSummaryCard.tsx
- SetupProgressIndicator.tsx

PHASE 4: NOTIFICATIONS & ENGAGEMENT

Prompt 4.1: Backend Notification Service

Create the notification service for check-in reminders and spending alerts.

CONTEXT:
- Flask backend
- Will use Firebase Cloud Messaging for push notifications (free tier)
- Will use existing email service for email reminders
- Celery for background task scheduling

CREATE: backend/services/notification_service.py

REQUIREMENTS:

1. NotificationService class:

schedule_weekly_reminders(user_id: str) -> None
- Schedule reminders for Sunday 6pm, Monday 10am, Monday 7pm
- Only if check-in not completed for current week

send_checkin_reminder(user_id: str, reminder_type: str) -> None
- reminder_type: 'first', 'second', 'streak_at_risk'
- Check if check-in already completed before sending
- Include spending tracking benefit in message

send_streak_at_risk_notification(user_id: str, current_streak: int) -> None
- Special urgent notification when streak > 2 and deadline approaching

send_insight_notification(user_id: str, insight_title: str, dollar_amount: float = None) -> None
- Notify when new significant insight is discovered
- Include dollar amount if applicable: "New insight: High stress weeks cost you $85 more"

send_spending_alert(user_id: str, category: str, amount: float, vs_average: float) -> None
- Alert when spending in a category is significantly above average
- "Heads up: Your shopping this week ($180) is 2x your average"

2. Notification templates:

```
TEMPLATES = {
  'checkin_first': {
    'title': "Time for your weekly check-in! 🌟 ",
    'body': "Track your wellness AND spending in 7 minutes.",
    'action': "Start Check-in"
  },
  'checkin_reminder': {...},
  'streak_at_risk': {
    'title': "🔥 {streak}-week streak at risk!",
    'body': "Complete your check-in to keep your streak alive.",
    'action': "Save My Streak"
  },
  'new_insight': {
    'title': "💡 New insight unlocked!",
    'body': "{insight_title}",
    'action': "See Insight"
  },
  'spending_alert': {
    'title': "📊 Spending heads-up",
    'body': "Your {category} this week (${amount}) is {percent}% above your average.",
    'action': "View Details"
  }
}
```

3. User preferences:
- Check user's notification preferences before sending
- Respect quiet hours (10pm-8am unless urgent)

- Allow users to disable specific notification types

4. Celery tasks in backend/tasks/notification_tasks.py:
   - check_and_send_reminders() - Runs hourly
   - send_scheduled_notification(user_id, notification_type)

Register in backend/api/__init__.py

Prompt 4.2: Streak and Gamification System

Create the streak tracking and gamification system.

CONTEXT:
- Flask backend with SQLAlchemy
- Encourages consistent weekly check-ins
- Celebrates spending wins, not just wellness wins

CREATE: backend/services/gamification_service.py

REQUIREMENTS:

1. StreakService class:

   update_streak(user_id: str, checkin_date: date) -> dict
   - Called after each check-in submission
   - Calculate if streak continues, breaks, or starts new
   - Return { current_streak, longest_streak, streak_increased: bool, new_record: bool }

   get_streak_info(user_id: str) -> dict
   - Return current streak info with days_until_deadline

   is_streak_at_risk(user_id: str) -> bool
   - True if streak > 0 and < 24 hours until week ends

2. AchievementService class:

   ACHIEVEMENTS = {
    # Check-in achievements
    'first_checkin': { 'name': 'Getting Started', 'icon': '🌱' },
    'streak_4': { 'name': 'Consistency Champion', 'icon': '🔥' },
    'streak_12': { 'name': 'Quarterly Commitment', 'icon': '💎' },

```
    # Wellness achievements
    'perfect_wellness_75': { 'name': 'Thriving', 'icon': '⭐' },
    'exercise_5_days': { 'name': 'Fitness Five', 'icon': '💪' },
    'meditation_streak_4': { 'name': 'Mindful Month', 'icon': '🧘' },

    # Spending achievements (NEW)
    'no_impulse_week': { 'name': 'Mindful Spending', 'description': 'Zero impulse purchases in a week', 'icon': '🎯' },
    'no_impulse_streak_4': { 'name': 'Impulse Control Master', 'description': '4 weeks with no impulse spending', 'icon': '🏆' },
    'below_average_week': { 'name': 'Under Budget', 'description': 'Total spending below your average', 'icon': '💚' },
    'below_average_streak_4': { 'name': 'Budget Boss', 'description': '4 weeks below average spending', 'icon': '👑' },
    'no_stress_spending': { 'name': 'Stress-Free Spender', 'description': 'No stress spending this week', 'icon': '😌' },
    'lowest_spending_week': { 'name': 'New Record!', 'description': 'Your lowest spending week', 'icon': '📉' },

    # Insight achievements
    'insight_unlocked': { 'name': 'Pattern Finder', 'icon': '🔍' },
    'all_correlations_found': { 'name': 'Self-Aware', 'icon': '🧠' },
  }

  check_achievements(user_id: str, checkin: dict, streak: int, baselines: dict) -> List[str]
  - Check all achievement conditions after check-in
  - Return list of newly unlocked achievement keys

  get_user_achievements(user_id: str) -> List[dict]
  - Return all achievements with unlocked status

3. Database table:
  CREATE TABLE user_achievements (...)

4. API endpoints:
  GET /api/wellness/achievements

Add migration for user_achievements table.
```

PHASE 5: TESTING

Prompt 5.1: Comprehensive Backend Tests

```
Create comprehensive tests for all Weekly Check-in System backend services.
```

CONTEXT:
- pytest for testing
- Tests in tests/ directory
- The system now includes spending estimates in check-ins

CREATE:

1. tests/test_wellness_score_service.py
   - Test all score calculations with edge cases
   - Test week-over-week change calculations
   - Test input validation errors

2. tests/test_correlation_engine_service.py
   - Test Pearson correlation calculation
   - Test with insufficient data (< 4 weeks)
   - Test each correlation type using check-in spending data
   - Test insight message generation with dollar amounts
   - Test handling of null spending values

3. tests/test_spending_baseline_service.py
   - Test baseline calculation with various data patterns
   - Test comparison to baseline (status assignment)
   - Test with missing/null spending categories
   - Test minimum data requirements

4. tests/test_insight_generator_service.py
   - Test spending pattern insights
   - Test anomaly detection with spending data
   - Test achievement detection for spending milestones
   - Test priority sorting

5. tests/test_wellness_api.py
   - Test POST /checkin with full wellness + spending data
   - Test spending history endpoint
   - Test baselines endpoint
   - Test partial spending data handling

6. tests/test_gamification_service.py
   - Test spending-related achievements
   - Test streak calculation

7. tests/fixtures/wellness_fixtures.py
   - Sample check-ins with spending data for 4, 8, 12 weeks
   - Sample baselines

- Users at various stages

Aim for > 90% code coverage.

Prompt 5.2: Frontend Component Tests

Create tests for all Weekly Check-in frontend components.

CONTEXT:
- Jest and React Testing Library
- Tests in __tests__ folders or .test.tsx files

CREATE:

1. WeeklyCheckinForm.test.tsx
   - Test all 6 steps including spending step
   - Test spending input validation
   - Test quick-estimate buttons
   - Test impulse/stress spending toggles
   - Test form submission with full data

2. WellnessScoreCard.test.tsx
   - Test score display and color coding
   - Test change indicators

3. SpendingSummaryCard.test.tsx
   - Test spending display
   - Test comparison to baselines
   - Test empty/insufficient data states
   - Test category highlighting for above-average

4. WellnessImpactCard.test.tsx
   - Test spending_pattern insight type
   - Test dollar amount display
   - Test various insight types

5. FinancialSetupFlow.test.tsx
   - Test income entry
   - Test recurring expense entry
   - Test summary calculation
   - Test skip functionality

6. hooks/useWellnessData.test.tsx
   - Test data fetching including spending
   - Test refetch after check-in

Use snapshot tests for UI consistency.
Test accessibility with jest-axe.

Prompt 5.3: End-to-End Test Suite

Create end-to-end tests for the complete check-in flow with spending.

CONTEXT:
- Cypress for E2E testing
- Tests in cypress/e2e/

CREATE: cypress/e2e/weekly-checkin-with-spending.cy.ts

TEST SCENARIOS:

1. First-time user complete flow
   - User creates account
   - Completes financial setup (income + recurring expenses)
   - Completes first weekly check-in including spending estimates
   - Sees wellness score and spending summary
   - Sees "X more weeks to unlock insights" message

2. Returning user with spending data
   - User with 3 previous check-ins logs in
   - Completes check-in with spending estimates
   - Sees spending comparison to their baseline
   - If week 4, sees first insights unlock

3. Spending insight generation
   - User with 5 weeks of data showing stress-spending correlation
   - Completes high-stress check-in with high impulse spending
   - Sees correlation insight generated
   - Sees anomaly alert for current week

4. Spending achievement unlock
   - User reports $0 impulse spending

- "Mindful Spending" achievement unlocks
- Achievement notification appears

5. Above-average spending alert
   - User reports shopping 2x their average
   - SpendingSummaryCard highlights this
   - Relevant insight/recommendation appears

Include fixtures for test users at different stages with spending history.

IMPLEMENTATION CHECKLIST

Phase 1: Database & Backend Foundation

- [ ] 1.1 Database schema (wellness + spending + financial setup)
- [ ] 1.2 Wellness score calculator service
- [ ] 1.3 Correlation engine service (using check-in spending)
- [ ] 1.4 Insight generator service (with spending patterns)
- [ ] 1.5 Weekly check-in API endpoints (with spending)
- [ ] 1.6 Financial setup API (recurring expenses + income)
- [ ] 1.7 Spending baseline calculator service

Phase 2: Frontend Implementation

- [ ] 2.1 Weekly check-in form (6 steps with spending)
- [ ] 2.2 Wellness score card component
- [ ] 2.3 Spending summary card component (NEW)
- [ ] 2.4 Wellness impact card component
- [ ] 2.5 Check-in reminder banner
- [ ] 2.6 Dashboard integration

Phase 3: Onboarding & Financial Setup

- [ ] 3.1 Financial setup flow component

Phase 4: Notifications & Engagement

- [ ] 4.1 Backend notification service
- [ ] 4.2 Streak and gamification system

Phase 5: Testing

- [ ] 5.1 Backend unit tests
- [ ] 5.2 Frontend component tests
- [ ] 5.3 End-to-end tests

KEY CHANGES FROM PREVIOUS VERSION

1. **Spending estimates are now part of the weekly check-in** (Step 5)
    - No separate expense tracking needed for MVP
    - No bank integration (Plaid) required
2. **New database fields** for spending in weekly_checkins table
    - groceries_estimate, dining_estimate, entertainment_estimate, etc.
    - impulse_spending, stress_spending (directly asked)
3. **New tables** for financial setup
    - recurring_expenses (one-time setup)
    - user_income (one-time setup)
    - user_spending_baselines (computed)
4. **Correlation engine** now uses check-in data directly
    - No need to query separate expense table
    - Correlations work with spending estimates
5. **New SpendingSummaryCard** component
    - Shows weekly spending at a glance
    - Compares to user's own baselines
6. **Spending-related achievements** added
    - "No impulse spending" milestones
    - "Under budget" achievements
7. **Financial Setup Flow** added to onboarding
    - One-time income and recurring expense setup
    - Provides context for variable spending tracking

Total estimated implementation time: 5-7 weeks with one developer External costs: ~$30/month (push notifications + email) No bank integration (Plaid) required for MVP