**Mingus User Screening Tool - Development Guide**

**Purpose & Objectives**

**Primary Goal**

Create a predictive screening tool that identifies users most likely to benefit from and engage with Mingus based on demographic characteristics, financial behaviors, and psychological patterns.

**Business Outcomes**

* **Reduce churn** by filtering out users unlikely to engage long-term
* **Increase conversion rates** by focusing marketing on high-probability users
* **Optimize onboarding** by tailoring experience to user segments
* **Improve product-market fit** by understanding core user characteristics

**Phase 1: Data Foundation & Analysis**

**Step 1: Define Success Metrics**

**Primary Success Indicators**

USER\_SUCCESS\_METRICS = {

'engagement': {

'daily\_active\_30\_days': 0.7, # 70% of days active in first month

'feature\_adoption': 0.8, # Uses 80%+ of core features

'health\_checkin\_completion': 0.9, # Completes 90%+ weekly check-ins

'financial\_data\_maintenance': 0.85 # Keeps financial data updated

},

'retention': {

'30\_day\_retention': 0.75, # 75% still active after 30 days

'90\_day\_retention': 0.60, # 60% still active after 90 days

'6\_month\_retention': 0.45 # 45% still active after 6 months

},

'value\_realization': {

'goal\_achievement': 0.50, # Achieves 50%+ of stated goals

'financial\_improvement': 0.40, # Shows measurable financial progress

'stress\_reduction': 0.35, # Reports reduced financial stress

'recommendation\_likelihood': 8.0 # NPS score of 8+ (0-10 scale)

}

}

**Failure/Churn Indicators**

CHURN\_RISK\_FACTORS = {

'early\_warning\_signs': [

'incomplete\_onboarding', # Doesn't complete financial profile

'low\_initial\_engagement', # <3 sessions in first week

'inconsistent\_data\_entry', # Sporadic check-in completion

'feature\_avoidance', # Avoids core features like cash flow

'support\_ticket\_frequency' # Multiple support requests early on

],

'demographic\_risk\_factors': [

'income\_instability', # Irregular income patterns

'excessive\_financial\_stress', # Stress levels consistently 8+/10

'minimal\_financial\_goals', # No clear financial objectives

'low\_digital\_literacy', # Struggles with app navigation

'privacy\_concerns' # Reluctant to share financial data

]

}

**Step 2: Historical Data Analysis**

**Data Collection Framework**

class UserDataAnalyzer:

def \_\_init\_\_(self):

self.success\_users = []

self.churned\_users = []

self.active\_users = []

def collect\_user\_cohorts(self):

"""

Analyze existing users (even small sample from beta/MVP)

"""

# Success cohort: High engagement + retention

success\_criteria = {

'sessions\_per\_week': '>= 4',

'feature\_usage': '>= 0.8',

'retention\_30\_day': True,

'financial\_data\_quality': 'complete'

}

# Churn cohort: Low engagement or stopped using

churn\_criteria = {

'last\_login': '> 14 days ago',

'incomplete\_onboarding': True,

'low\_engagement': '< 2 sessions/week',

'support\_issues': '> 2 tickets'

}

return self.segment\_users(success\_criteria, churn\_criteria)

def identify\_patterns(self, user\_cohorts):

"""

Find distinguishing characteristics between success/churn cohorts

"""

patterns = {

'demographic\_patterns': self.analyze\_demographics(user\_cohorts),

'behavioral\_patterns': self.analyze\_behaviors(user\_cohorts),

'psychographic\_patterns': self.analyze\_attitudes(user\_cohorts),

'financial\_patterns': self.analyze\_financial\_health(user\_cohorts)

}

return patterns

**Step 3: Questionnaire Design**

**Core Screening Questions (5-7 minutes max)**

**Section A: Demographics & Context**

DEMOGRAPHIC\_QUESTIONS = [

{

'id': 'age\_range',

'question': 'What\'s your age range?',

'type': 'single\_choice',

'options': ['18-24', '25-29', '30-35', '36-40', '40+'],

'weight': 0.15,

'success\_correlation': {

'25-29': 0.85, # Highest success rate

'30-35': 0.90, # Peak target demographic

'18-24': 0.60, # Lower engagement

'36-40': 0.70, # Moderate success

'40+': 0.50 # Lowest success rate

}

},

{

'id': 'employment\_status',

'question': 'What best describes your current employment?',

'type': 'single\_choice',

'options': [

'Full-time employee',

'Part-time employee',

'Freelancer/Contractor',

'Entrepreneur/Business owner',

'Between jobs',

'Student'

],

'weight': 0.20,

'success\_correlation': {

'Full-time employee': 0.85,

'Freelancer/Contractor': 0.75,

'Entrepreneur/Business owner': 0.80,

'Part-time employee': 0.60,

'Between jobs': 0.45,

'Student': 0.40

}

},

{

'id': 'income\_stability',

'question': 'How predictable is your monthly income?',

'type': 'scale',

'scale': '1-10 (1=Very unpredictable, 10=Very predictable)',

'weight': 0.25,

'success\_correlation': 'linear\_positive' # Higher stability = higher success

}

]

**Section B: Financial Behavior Patterns**

FINANCIAL\_BEHAVIOR\_QUESTIONS = [

{

'id': 'budgeting\_experience',

'question': 'How would you describe your budgeting experience?',

'type': 'single\_choice',

'options': [

'I\'ve never really budgeted',

'I\'ve tried but struggled to stick with it',

'I budget occasionally when needed',

'I budget regularly and track expenses',

'I\'m very disciplined with detailed budgeting'

],

'weight': 0.20,

'success\_correlation': {

'I\'ve tried but struggled to stick with it': 0.90, # Perfect fit

'I budget occasionally when needed': 0.85,

'I\'ve never really budgeted': 0.75,

'I budget regularly and track expenses': 0.70,

'I\'m very disciplined with detailed budgeting': 0.50 # May not need app

}

},

{

'id': 'financial\_stress\_frequency',

'question': 'How often do you worry about money?',

'type': 'single\_choice',

'options': [

'Daily',

'Several times a week',

'Weekly',

'Monthly',

'Rarely',

'Never'

],

'weight': 0.25,

'success\_correlation': {

'Several times a week': 0.95, # Sweet spot - motivated but not overwhelmed

'Weekly': 0.85,

'Daily': 0.75, # May be too stressed to engage

'Monthly': 0.65,

'Rarely': 0.40,

'Never': 0.20 # May not see value

}

},

{

'id': 'financial\_goals\_clarity',

'question': 'How clear are you about your financial goals?',

'type': 'scale',

'scale': '1-10 (1=No clear goals, 10=Very specific goals)',

'weight': 0.15,

'success\_correlation': 'moderate\_positive' # Too low or too high both problematic

}

]

**Section C: Technology & App Usage**

TECHNOLOGY\_QUESTIONS = [

{

'id': 'fintech\_app\_experience',

'question': 'Which financial apps have you used before? (Select all)',

'type': 'multiple\_choice',

'options': [

'Mint',

'YNAB (You Need A Budget)',

'Personal Capital',

'Acorns',

'Cash App',

'Venmo',

'Banking apps only',

'None of the above'

],

'weight': 0.15,

'success\_correlation': {

'Mint': 0.70, # May be looking for something better

'YNAB (You Need A Budget)': 0.60, # Already have solution

'Cash App': 0.85, # Good tech comfort, simpler apps

'Banking apps only': 0.80, # Basic comfort, room for growth

'None of the above': 0.75 # Open to new solutions

}

},

{

'id': 'data\_sharing\_comfort',

'question': 'How comfortable are you sharing financial information with a secure app?',

'type': 'scale',

'scale': '1-10 (1=Very uncomfortable, 10=Very comfortable)',

'weight': 0.20,

'success\_correlation': 'threshold\_6' # Need at least 6/10 for success

}

]

**Section D: Lifestyle & Values**

LIFESTYLE\_QUESTIONS = [

{

'id': 'health\_wellness\_priority',

'question': 'How important is health and wellness in your daily life?',

'type': 'scale',

'scale': '1-10 (1=Not important, 10=Very important)',

'weight': 0.20,

'success\_correlation': 'linear\_positive' # Core differentiator for Mingus

},

{

'id': 'stress\_management\_interest',

'question': 'Are you interested in understanding how stress affects your spending?',

'type': 'single\_choice',

'options': [

'Very interested - I notice this connection',

'Somewhat interested - curious to learn more',

'Neutral - might be helpful',

'Not very interested',

'Not interested at all'

],

'weight': 0.25,

'success\_correlation': {

'Very interested - I notice this connection': 0.95,

'Somewhat interested - curious to learn more': 0.80,

'Neutral - might be helpful': 0.50,

'Not very interested': 0.25,

'Not interested at all': 0.10

}

},

{

'id': 'community\_connection',

'question': 'How important is it to you that financial advice considers your cultural background?',

'type': 'scale',

'scale': '1-10 (1=Not important, 10=Very important)',

'weight': 0.15,

'success\_correlation': 'moderate\_positive' # Mingus differentiator

}

]

**Phase 2: Scoring Algorithm Development**

**Step 4: Predictive Scoring Model**

**Multi-Factor Success Probability Calculator**

class MingusScreeningScorer:

def \_\_init\_\_(self):

self.weights = {

'demographic\_fit': 0.25,

'financial\_readiness': 0.30,

'technology\_comfort': 0.20,

'value\_alignment': 0.25

}

self.thresholds = {

'high\_success': 0.75, # 75%+ probability of success

'medium\_success': 0.60, # 60-74% probability

'low\_success': 0.45, # 45-59% probability

'poor\_fit': 0.44 # <45% probability

}

def calculate\_success\_probability(self, responses):

"""

Calculate overall success probability based on questionnaire responses

"""

scores = {

'demographic\_fit': self.score\_demographics(responses),

'financial\_readiness': self.score\_financial\_readiness(responses),

'technology\_comfort': self.score\_technology\_comfort(responses),

'value\_alignment': self.score\_value\_alignment(responses)

}

# Weighted average

overall\_score = sum(

scores[factor] \* self.weights[factor]

for factor in scores

)

return {

'overall\_probability': overall\_score,

'component\_scores': scores,

'recommendation': self.get\_recommendation(overall\_score),

'personalization\_flags': self.get\_personalization\_flags(responses)

}

def score\_demographics(self, responses):

"""Score based on demographic alignment with successful users"""

score = 0.0

# Age alignment

age\_scores = {

'25-29': 0.85, '30-35': 0.90, '18-24': 0.60,

'36-40': 0.70, '40+': 0.50

}

score += age\_scores.get(responses.get('age\_range'), 0.5) \* 0.4

# Employment stability

employment\_scores = {

'Full-time employee': 0.85, 'Freelancer/Contractor': 0.75,

'Entrepreneur/Business owner': 0.80, 'Part-time employee': 0.60,

'Between jobs': 0.45, 'Student': 0.40

}

score += employment\_scores.get(responses.get('employment\_status'), 0.5) \* 0.6

return min(1.0, score)

def score\_financial\_readiness(self, responses):

"""Score financial behavior patterns and readiness"""

score = 0.0

# Budgeting experience (sweet spot: tried but struggled)

budgeting\_scores = {

'I\'ve tried but struggled to stick with it': 0.90,

'I budget occasionally when needed': 0.85,

'I\'ve never really budgeted': 0.75,

'I budget regularly and track expenses': 0.70,

'I\'m very disciplined with detailed budgeting': 0.50

}

score += budgeting\_scores.get(responses.get('budgeting\_experience'), 0.5) \* 0.3

# Financial stress frequency (sweet spot: several times a week)

stress\_scores = {

'Several times a week': 0.95, 'Weekly': 0.85, 'Daily': 0.75,

'Monthly': 0.65, 'Rarely': 0.40, 'Never': 0.20

}

score += stress\_scores.get(responses.get('financial\_stress\_frequency'), 0.5) \* 0.4

# Income stability (linear positive correlation)

income\_stability = int(responses.get('income\_stability', 5))

score += (income\_stability / 10.0) \* 0.3

return min(1.0, score)

def score\_technology\_comfort(self, responses):

"""Score technology adoption and comfort levels"""

score = 0.0

# Data sharing comfort (threshold-based)

data\_comfort = int(responses.get('data\_sharing\_comfort', 5))

if data\_comfort >= 6:

score += 0.6

elif data\_comfort >= 4:

score += 0.4

else:

score += 0.1

# Fintech experience

fintech\_apps = responses.get('fintech\_app\_experience', [])

if isinstance(fintech\_apps, str):

fintech\_apps = [fintech\_apps]

experience\_score = 0.5 # Default

if 'Cash App' in fintech\_apps or 'Banking apps only' in fintech\_apps:

experience\_score = 0.8

elif 'None of the above' in fintech\_apps:

experience\_score = 0.75

elif 'Mint' in fintech\_apps:

experience\_score = 0.7

elif 'YNAB (You Need A Budget)' in fintech\_apps:

experience\_score = 0.6

score += experience\_score \* 0.4

return min(1.0, score)

def score\_value\_alignment(self, responses):

"""Score alignment with Mingus's unique value propositions"""

score = 0.0

# Health-wellness priority

health\_priority = int(responses.get('health\_wellness\_priority', 5))

score += (health\_priority / 10.0) \* 0.4

# Interest in stress-spending connection

stress\_interest\_scores = {

'Very interested - I notice this connection': 0.95,

'Somewhat interested - curious to learn more': 0.80,

'Neutral - might be helpful': 0.50,

'Not very interested': 0.25,

'Not interested at all': 0.10

}

score += stress\_interest\_scores.get(responses.get('stress\_management\_interest'), 0.5) \* 0.4

# Cultural consideration importance

cultural\_importance = int(responses.get('community\_connection', 5))

score += (cultural\_importance / 10.0) \* 0.2

return min(1.0, score)

def get\_recommendation(self, score):

"""Get recommendation based on overall score"""

if score >= self.thresholds['high\_success']:

return {

'category': 'high\_success',

'action': 'immediate\_onboarding',

'message': 'Great fit! You\'ll love Mingus\'s holistic approach to financial wellness.',

'priority': 'high',

'onboarding\_track': 'full\_experience'

}

elif score >= self.thresholds['medium\_success']:

return {

'category': 'medium\_success',

'action': 'targeted\_onboarding',

'message': 'Mingus could be a great fit with the right approach.',

'priority': 'medium',

'onboarding\_track': 'guided\_experience'

}

elif score >= self.thresholds['low\_success']:

return {

'category': 'low\_success',

'action': 'educational\_nurture',

'message': 'Learn more about how Mingus can help your financial journey.',

'priority': 'low',

'onboarding\_track': 'educational\_sequence'

}

else:

return {

'category': 'poor\_fit',

'action': 'alternative\_resources',

'message': 'Mingus might not be the right fit right now.',

'priority': 'nurture',

'onboarding\_track': 'resource\_sharing'

}

**Phase 3: Implementation & Integration**

**Step 5: Frontend Implementation**

**Screening Tool Interface**

// React component for screening questionnaire

class MingusScreeningTool extends React.Component {

constructor(props) {

super(props);

this.state = {

currentSection: 0,

responses: {},

isLoading: false,

results: null

};

this.sections = [

{ title: 'About You', questions: DEMOGRAPHIC\_QUESTIONS },

{ title: 'Your Money Story', questions: FINANCIAL\_BEHAVIOR\_QUESTIONS },

{ title: 'Tech Comfort', questions: TECHNOLOGY\_QUESTIONS },

{ title: 'Your Values', questions: LIFESTYLE\_QUESTIONS }

];

}

async submitScreening() {

this.setState({ isLoading: true });

try {

const response = await fetch('/api/screening/evaluate', {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify({

responses: this.state.responses,

timestamp: Date.now(),

source: 'landing\_page'

})

});

const results = await response.json();

this.setState({ results, isLoading: false });

// Track screening completion

this.trackScreeningCompletion(results);

} catch (error) {

console.error('Screening submission failed:', error);

this.setState({ isLoading: false });

}

}

renderResults() {

const { results } = this.state;

if (!results) return null;

return (

<div className="screening-results">

<div className="compatibility-score">

<h2>Your Mingus Compatibility</h2>

<div className="score-circle">

<span className="score-value">

{Math.round(results.overall\_probability \* 100)}%

</span>

<span className="score-label">Match</span>

</div>

</div>

<div className="recommendation">

<h3>{results.recommendation.message}</h3>

{this.renderActionButton(results.recommendation)}

</div>

<div className="score-breakdown">

<h4>How We Calculated This</h4>

{Object.entries(results.component\_scores).map(([factor, score]) => (

<div key={factor} className="score-component">

<span className="factor-name">{this.formatFactorName(factor)}</span>

<div className="score-bar">

<div

className="score-fill"

style={{ width: `${score \* 100}%` }}

></div>

</div>

<span className="score-percent">{Math.round(score \* 100)}%</span>

</div>

))}

</div>

</div>

);

}

renderActionButton(recommendation) {

switch (recommendation.action) {

case 'immediate\_onboarding':

return (

<button

className="cta-primary large"

onClick={() => this.redirectToOnboarding('full\_experience')}

>

Start Your Free Trial

</button>

);

case 'targeted\_onboarding':

return (

<button

className="cta-secondary large"

onClick={() => this.redirectToOnboarding('guided\_experience')}

>

Learn More & Try Mingus

</button>

);

case 'educational\_nurture':

return (

<button

className="cta-secondary"

onClick={() => this.showEducationalContent()}

>

Get Free Financial Wellness Guide

</button>

);

default:

return (

<button

className="cta-tertiary"

onClick={() => this.showAlternativeResources()}

>

Explore Other Resources

</button>

);

}

}

}

**Step 6: Backend Implementation**

**Screening API Endpoint**

# views.py

from django.http import JsonResponse

from .screening\_scorer import MingusScreeningScorer

from .models import ScreeningResponse

@require\_http\_methods(["POST"])

def evaluate\_screening(request):

"""

Evaluate screening questionnaire and return personalized recommendations

"""

try:

data = json.loads(request.body)

responses = data.get('responses', {})

# Validate responses

if not responses:

return JsonResponse({'error': 'No responses provided'}, status=400)

# Calculate success probability

scorer = MingusScreeningScorer()

results = scorer.calculate\_success\_probability(responses)

# Save screening response for analysis

screening\_response = ScreeningResponse.objects.create(

responses=responses,

overall\_score=results['overall\_probability'],

recommendation\_category=results['recommendation']['category'],

source=data.get('source', 'unknown'),

ip\_address=get\_client\_ip(request),

user\_agent=request.META.get('HTTP\_USER\_AGENT', '')

)

# Track for analytics

track\_screening\_completion(screening\_response, results)

return JsonResponse({

'success': True,

'screening\_id': screening\_response.id,

\*\*results

})

except Exception as e:

logger.error(f"Screening evaluation failed: {e}")

return JsonResponse({'error': 'Evaluation failed'}, status=500)

def track\_screening\_completion(screening\_response, results):

"""Track screening completion for analytics"""

analytics\_data = {

'event': 'screening\_completed',

'screening\_id': screening\_response.id,

'overall\_score': results['overall\_probability'],

'recommendation': results['recommendation']['category'],

'component\_scores': results['component\_scores']

}

# Send to analytics platform

send\_to\_analytics(analytics\_data)

**Phase 4: Optimization & Refinement**

**Step 7: A/B Testing Framework**

**Testing Different Scoring Models**

class ScreeningExperimentManager:

def \_\_init\_\_(self):

self.experiments = {

'scoring\_model\_v1': {

'weights': {

'demographic\_fit': 0.25,

'financial\_readiness': 0.30,

'technology\_comfort': 0.20,

'value\_alignment': 0.25

},

'active': True,

'traffic\_percentage': 50

},

'scoring\_model\_v2': {

'weights': {

'demographic\_fit': 0.20,

'financial\_readiness': 0.35,

'technology\_comfort': 0.15,

'value\_alignment': 0.30

},

'active': True,

'traffic\_percentage': 50

}

}

def get\_scoring\_model(self, user\_id):

"""Assign user to experiment group"""

experiment\_hash = hash(f"{user\_id}\_screening") % 100

if experiment\_hash < 50:

return 'scoring\_model\_v1'

else:

return 'scoring\_model\_v2'

def track\_outcomes(self, screening\_id, actual\_user\_success):

"""Track actual user outcomes vs. predictions"""

screening = ScreeningResponse.objects.get(id=screening\_id)

# Update with actual outcome

screening.actual\_30\_day\_retention = actual\_user\_success['retained\_30\_days']

screening.actual\_engagement\_score = actual\_user\_success['engagement\_score']

screening.save()

# Calculate prediction accuracy

predicted\_success = screening.overall\_score > 0.6

actual\_success = actual\_user\_success['retained\_30\_days'] and actual\_user\_success['engagement\_score'] > 0.7

# Track accuracy for model optimization

ModelAccuracyLog.objects.create(

screening\_response=screening,

predicted\_success=predicted\_success,

actual\_success=actual\_success,

accuracy\_score=self.calculate\_accuracy(predicted\_success, actual\_success)

)

**Step 8: Continuous Improvement**

**Monthly Model Refinement**

class ScreeningModelOptimizer:

def analyze\_prediction\_accuracy(self, time\_period\_days=30):

"""

Analyze how well the screening tool predicts actual user success

"""

cutoff\_date = timezone.now() - timedelta(days=time\_period\_days)

recent\_screenings = ScreeningResponse.objects.filter(

created\_at\_\_gte=cutoff\_date,

actual\_30\_day\_retention\_\_isnull=False # Has actual outcome data

)

accuracy\_metrics = {

'total\_predictions': recent\_screenings.count(),

'correct\_predictions': 0,

'false\_positives': 0, # Predicted success, actually churned

'false\_negatives': 0, # Predicted churn, actually succeeded

'precision': 0.0,

'recall': 0.0,

'f1\_score': 0.0

}

for screening in recent\_screenings:

predicted\_success = screening.overall\_score > 0.6

actual\_success = (screening.actual\_30\_day\_retention and

screening.actual\_engagement\_score > 0.7)

if predicted\_success == actual\_success:

accuracy\_metrics['correct\_predictions'] += 1

elif predicted\_success and not actual\_success:

accuracy\_metrics['false\_positives'] += 1

elif not predicted\_success and actual\_success:

accuracy\_metrics['false\_negatives'] += 1

# Calculate precision, recall, F1

if accuracy\_metrics['total\_predictions'] > 0:

accuracy = accuracy\_metrics['correct\_predictions'] / accuracy\_metrics['total\_predictions']

accuracy\_metrics['overall\_accuracy'] = accuracy

return accuracy\_metrics

def recommend\_model\_adjustments(self, accuracy\_metrics):

"""

Recommend adjustments based on prediction accuracy

"""

recommendations = []

if accuracy\_metrics['false\_positives'] > accuracy\_metrics['false\_negatives']:

recommendations.append("Model is too optimistic - consider raising success thresholds")

elif accuracy\_metrics['false\_negatives'] > accuracy\_metrics['false\_positives']:

recommendations.append("Model is too pessimistic - consider lowering success thresholds")

if accuracy\_metrics['overall\_accuracy'] < 0.7:

recommendations.append("Overall accuracy below 70% - review question weights and scoring logic")

return recommendations

**Phase 5: Launch & Marketing Integration**

**Step 9: Marketing Funnel Integration**

**Lead Scoring for Marketing**

def integrate\_with\_marketing\_automation(screening\_results, user\_email):

"""

Send screening results to marketing automation platform

"""

marketing\_data = {

'email': user\_email,

'lead\_score': screening\_results['overall\_probability'] \* 100,

'segment': screening\_results['recommendation']['category'],

'personalization\_flags': screening\_results['personalization\_flags'],

'recommended\_onboarding\_track': screening\_results['recommendation']['onboarding\_track']

}

# Send to marketing platform (e.g., HubSpot, Mailchimp)

marketing\_api.update\_contact(marketing\_data)

# Trigger appropriate email sequence

if marketing\_data['segment'] == 'high\_success':

trigger\_email\_sequence('immediate\_trial\_sequence', user\_email)

elif marketing\_data['segment'] == 'medium\_success':

trigger\_email\_sequence('nurture\_sequence', user\_email)

elif marketing\_data['segment'] == 'low\_success':

trigger\_email\_sequence('educational\_sequence', user\_email)

else:

trigger\_email\_sequence('alternative\_resources\_sequence', user\_email)

#### \*\*Personalized Onboarding Tracks\*\*

```python

class PersonalizedOnboardingManager:

def \_\_init\_\_(self):

self.onboarding\_tracks = {

'full\_experience': {

'description': 'Complete onboarding for high-probability users',

'steps': [

'welcome\_video\_targeted',

'financial\_profile\_full',

'health\_integration\_emphasis',

'job\_security\_setup',

'milestone\_planning',

'dashboard\_tour\_complete'

],

'estimated\_time': '15-20 minutes',

'skip\_options': False

},

'guided\_experience': {

'description': 'Simplified onboarding with extra guidance',

'steps': [

'welcome\_video\_educational',

'financial\_profile\_simplified',

'health\_integration\_intro',

'basic\_goal\_setting',

'dashboard\_tour\_guided'

],

'estimated\_time': '10-12 minutes',

'skip\_options': True

},

'educational\_sequence': {

'description': 'Education-focused with gradual feature introduction',

'steps': [

'financial\_wellness\_education',

'basic\_profile\_setup',

'feature\_preview',

'trial\_encouragement'

],

'estimated\_time': '8-10 minutes',

'skip\_options': True

},

'resource\_sharing': {

'description': 'Focus on educational resources rather than app features',

'steps': [

'financial\_wellness\_guide\_download',

'resource\_library\_access',

'newsletter\_signup',

'community\_access'

],

'estimated\_time': '3-5 minutes',

'skip\_options': True

}

}

def customize\_onboarding\_flow(self, screening\_results, user\_profile):

"""

Customize onboarding based on screening results and user characteristics

"""

track = screening\_results['recommendation']['onboarding\_track']

base\_flow = self.onboarding\_tracks[track].copy()

# Customize based on specific screening responses

customizations = self.get\_personalization\_customizations(

screening\_results['personalization\_flags'],

user\_profile

)

# Apply customizations

for customization in customizations:

base\_flow = self.apply\_customization(base\_flow, customization)

return base\_flow

def get\_personalization\_customizations(self, flags, user\_profile):

"""

Generate specific customizations based on user characteristics

"""

customizations = []

# Customize based on financial experience

if flags.get('budgeting\_experience') == 'none':

customizations.append({

'type': 'add\_step',

'step': 'budgeting\_basics\_tutorial',

'position': 'after\_financial\_profile'

})

# Customize based on technology comfort

if flags.get('tech\_comfort\_level') == 'low':

customizations.append({

'type': 'modify\_step',

'target': 'dashboard\_tour',

'modification': 'extend\_with\_detailed\_explanations'

})

# Customize based on health-finance interest

if flags.get('health\_finance\_interest') == 'high':

customizations.append({

'type': 'emphasize\_feature',

'feature': 'health\_correlation',

'emphasis\_level': 'high'

})

# Customize based on stress levels

if flags.get('financial\_stress\_level') == 'high':

customizations.append({

'type': 'add\_reassurance',

'message': 'stress\_reduction\_focus',

'placement': 'throughout\_flow'

})

return customizations

#### \*\*Dynamic Content Personalization\*\*

```python

class ScreeningContentPersonalizer:

def \_\_init\_\_(self):

self.content\_variants = {

'welcome\_messages': {

'high\_stress\_high\_interest': {

'headline': 'Finally, Relief for Your Money Stress',

'subheadline': 'Discover how your stress patterns affect spending and take back control',

'tone': 'empathetic\_solution\_focused'

},

'low\_experience\_high\_motivation': {

'headline': 'Your Personal Finance Journey Starts Here',

'subheadline': 'We\'ll guide you step-by-step to financial confidence',

'tone': 'encouraging\_educational'

},

'experienced\_looking\_for\_more': {

'headline': 'Level Up Your Financial Game',

'subheadline': 'Advanced insights that connect your whole life to your money',

'tone': 'sophisticated\_innovative'

}

},

'feature\_emphasis': {

'health\_focused': ['health\_correlation', 'stress\_tracking', 'wellness\_integration'],

'goal\_oriented': ['milestone\_planning', 'goal\_tracking', 'progress\_visualization'],

'security\_focused': ['job\_security', 'emergency\_planning', 'risk\_assessment'],

'simplicity\_focused': ['easy\_setup', 'automated\_insights', 'simple\_interface']

}

}

def generate\_personalized\_content(self, screening\_results, content\_type):

"""

Generate personalized content based on screening results

"""

flags = screening\_results['personalization\_flags']

category = screening\_results['recommendation']['category']

if content\_type == 'welcome\_message':

return self.select\_welcome\_message(flags, category)

elif content\_type == 'feature\_highlights':

return self.select\_feature\_highlights(flags)

elif content\_type == 'onboarding\_messaging':

return self.generate\_onboarding\_messages(flags, category)

def select\_welcome\_message(self, flags, category):

"""Select appropriate welcome message variant"""

stress\_level = flags.get('financial\_stress\_level', 'medium')

experience = flags.get('budgeting\_experience', 'medium')

health\_interest = flags.get('health\_finance\_interest', 'medium')

if stress\_level == 'high' and health\_interest == 'high':

return self.content\_variants['welcome\_messages']['high\_stress\_high\_interest']

elif experience == 'low' and category in ['high\_success', 'medium\_success']:

return self.content\_variants['welcome\_messages']['low\_experience\_high\_motivation']

elif experience == 'high':

return self.content\_variants['welcome\_messages']['experienced\_looking\_for\_more']

else:

# Default message

return {

'headline': 'Welcome to Mingus',

'subheadline': 'Your complete financial wellness companion',

'tone': 'friendly\_professional'

}

## Phase 6: Analytics & Measurement

### \*\*Step 10: Screening Analytics Dashboard\*\*

#### \*\*Screening Performance Metrics\*\*

```python

class ScreeningAnalyticsDashboard:

def \_\_init\_\_(self):

self.key\_metrics = [

'completion\_rate',

'prediction\_accuracy',

'conversion\_by\_segment',

'user\_satisfaction',

'business\_impact'

]

def generate\_screening\_analytics(self, date\_range):

"""

Generate comprehensive analytics for screening tool performance

"""

analytics = {

'overview': self.get\_overview\_metrics(date\_range),

'segment\_performance': self.analyze\_segment\_performance(date\_range),

'prediction\_accuracy': self.calculate\_prediction\_accuracy(date\_range),

'conversion\_funnel': self.analyze\_conversion\_funnel(date\_range),

'user\_feedback': self.compile\_user\_feedback(date\_range),

'business\_impact': self.calculate\_business\_impact(date\_range)

}

return analytics

def get\_overview\_metrics(self, date\_range):

"""Basic screening tool metrics"""

screenings = ScreeningResponse.objects.filter(

created\_at\_\_range=date\_range

)

completed\_screenings = screenings.filter(

responses\_\_isnull=False

).count()

started\_screenings = ScreeningStart.objects.filter(

created\_at\_\_range=date\_range

).count()

return {

'total\_screenings\_started': started\_screenings,

'total\_screenings\_completed': completed\_screenings,

'completion\_rate': completed\_screenings / started\_screenings if started\_screenings > 0 else 0,

'average\_completion\_time': self.calculate\_average\_completion\_time(screenings),

'segment\_distribution': self.get\_segment\_distribution(screenings)

}

def analyze\_segment\_performance(self, date\_range):

"""Analyze performance by user segment"""

segments = ['high\_success', 'medium\_success', 'low\_success', 'poor\_fit']

performance = {}

for segment in segments:

segment\_users = ScreeningResponse.objects.filter(

created\_at\_\_range=date\_range,

recommendation\_category=segment

)

# Calculate conversion rates

total\_users = segment\_users.count()

trial\_signups = segment\_users.filter(

user\_\_trial\_started\_\_isnull=False

).count()

active\_30\_days = segment\_users.filter(

actual\_30\_day\_retention=True

).count()

performance[segment] = {

'total\_users': total\_users,

'trial\_conversion\_rate': trial\_signups / total\_users if total\_users > 0 else 0,

'retention\_30\_day': active\_30\_days / total\_users if total\_users > 0 else 0,

'predicted\_vs\_actual': self.compare\_predicted\_vs\_actual(segment\_users)

}

return performance

def calculate\_prediction\_accuracy(self, date\_range):

"""Calculate how accurate our predictions are"""

screenings\_with\_outcomes = ScreeningResponse.objects.filter(

created\_at\_\_range=date\_range,

actual\_30\_day\_retention\_\_isnull=False,

actual\_engagement\_score\_\_isnull=False

)

accuracy\_data = {

'total\_predictions': screenings\_with\_outcomes.count(),

'correct\_predictions': 0,

'accuracy\_by\_segment': {},

'confusion\_matrix': {

'true\_positive': 0, # Predicted success, actually succeeded

'false\_positive': 0, # Predicted success, actually failed

'true\_negative': 0, # Predicted failure, actually failed

'false\_negative': 0 # Predicted failure, actually succeeded

}

}

for screening in screenings\_with\_outcomes:

predicted\_success = screening.overall\_score >= 0.6

actual\_success = (screening.actual\_30\_day\_retention and

screening.actual\_engagement\_score >= 0.7)

# Update confusion matrix

if predicted\_success and actual\_success:

accuracy\_data['confusion\_matrix']['true\_positive'] += 1

accuracy\_data['correct\_predictions'] += 1

elif predicted\_success and not actual\_success:

accuracy\_data['confusion\_matrix']['false\_positive'] += 1

elif not predicted\_success and not actual\_success:

accuracy\_data['confusion\_matrix']['true\_negative'] += 1

accuracy\_data['correct\_predictions'] += 1

else: # not predicted\_success and actual\_success

accuracy\_data['confusion\_matrix']['false\_negative'] += 1

# Calculate overall accuracy

total = accuracy\_data['total\_predictions']

if total > 0:

accuracy\_data['overall\_accuracy'] = accuracy\_data['correct\_predictions'] / total

# Calculate precision and recall

tp = accuracy\_data['confusion\_matrix']['true\_positive']

fp = accuracy\_data['confusion\_matrix']['false\_positive']

fn = accuracy\_data['confusion\_matrix']['false\_negative']

accuracy\_data['precision'] = tp / (tp + fp) if (tp + fp) > 0 else 0

accuracy\_data['recall'] = tp / (tp + fn) if (tp + fn) > 0 else 0

# F1 Score

p, r = accuracy\_data['precision'], accuracy\_data['recall']

accuracy\_data['f1\_score'] = 2 \* (p \* r) / (p + r) if (p + r) > 0 else 0

return accuracy\_data

### \*\*Step 11: Continuous Optimization Framework\*\*

#### \*\*Weekly Screening Review Process\*\*

```python

class ScreeningOptimizationEngine:

def \_\_init\_\_(self):

self.optimization\_schedule = {

'daily': ['monitor\_completion\_rates', 'track\_conversion\_anomalies'],

'weekly': ['analyze\_segment\_performance', 'review\_prediction\_accuracy'],

'monthly': ['optimize\_question\_weights', 'test\_new\_questions'],

'quarterly': ['comprehensive\_model\_review', 'competitive\_analysis']

}

def weekly\_optimization\_review(self):

"""

Weekly review and optimization recommendations

"""

end\_date = timezone.now()

start\_date = end\_date - timedelta(days=7)

# Gather performance data

analytics = ScreeningAnalyticsDashboard().generate\_screening\_analytics(

(start\_date, end\_date)

)

# Generate recommendations

recommendations = []

# Check completion rate

if analytics['overview']['completion\_rate'] < 0.7:

recommendations.append({

'priority': 'high',

'category': 'user\_experience',

'issue': 'Low completion rate',

'recommendation': 'Review questionnaire length and question clarity',

'suggested\_actions': [

'A/B test shorter questionnaire version',

'Analyze drop-off points',

'Simplify complex questions'

]

})

# Check prediction accuracy

if analytics['prediction\_accuracy']['overall\_accuracy'] < 0.75:

recommendations.append({

'priority': 'medium',

'category': 'model\_accuracy',

'issue': 'Prediction accuracy below target',

'recommendation': 'Adjust scoring weights or add new questions',

'suggested\_actions': [

'Analyze false positive/negative patterns',

'Review question weights',

'Consider adding behavioral questions'

]

})

# Check segment conversion rates

segment\_performance = analytics['segment\_performance']

for segment, data in segment\_performance.items():

if segment == 'high\_success' and data['trial\_conversion\_rate'] < 0.8:

recommendations.append({

'priority': 'high',

'category': 'conversion\_optimization',

'issue': f'Low conversion in {segment} segment',

'recommendation': 'Optimize post-screening messaging and onboarding',

'suggested\_actions': [

'Review onboarding flow for high-success users',

'Test different call-to-action messages',

'Analyze user feedback from this segment'

]

})

return {

'review\_date': end\_date,

'period\_analyzed': f"{start\_date.date()} to {end\_date.date()}",

'overall\_health': self.calculate\_overall\_health\_score(analytics),

'recommendations': recommendations,

'next\_review\_date': end\_date + timedelta(days=7)

}

def calculate\_overall\_health\_score(self, analytics):

"""

Calculate overall health score for screening tool

"""

weights = {

'completion\_rate': 0.3,

'prediction\_accuracy': 0.4,

'conversion\_rate': 0.3

}

scores = {

'completion\_rate': min(1.0, analytics['overview']['completion\_rate'] / 0.8),

'prediction\_accuracy': analytics['prediction\_accuracy']['overall\_accuracy'],

'conversion\_rate': min(1.0, analytics['segment\_performance']['high\_success']['trial\_conversion\_rate'] / 0.8)

}

overall\_score = sum(scores[metric] \* weights[metric] for metric in scores)

return {

'overall\_score': overall\_score,

'component\_scores': scores,

'health\_status': self.get\_health\_status(overall\_score)

}

def get\_health\_status(self, score):

"""Convert score to health status"""

if score >= 0.9:

return 'excellent'

elif score >= 0.8:

return 'good'

elif score >= 0.7:

return 'fair'

else:

return 'needs\_attention'

## Phase 7: Advanced Features & Future Enhancements

### \*\*Step 12: Advanced Screening Features\*\*

#### \*\*Dynamic Question Selection\*\*

```python

class AdaptiveScreeningEngine:

def \_\_init\_\_(self):

self.question\_bank = {

'core\_questions': [

# Always asked

'age\_range', 'employment\_status', 'financial\_stress\_frequency'

],

'conditional\_questions': {

'high\_stress\_detected': [

'stress\_coping\_mechanisms',

'financial\_trauma\_history',

'support\_system\_availability'

],

'entrepreneur\_detected': [

'business\_revenue\_stability',

'business\_financial\_separation',

'entrepreneur\_stress\_factors'

],

'high\_tech\_comfort': [

'advanced\_financial\_tools\_interest',

'data\_analysis\_comfort',

'automation\_preferences'

]

},

'clarification\_questions': {

'ambiguous\_financial\_stress': [

'stress\_source\_clarification',

'stress\_impact\_on\_decisions'

],

'unclear\_tech\_comfort': [

'specific\_app\_experience',

'learning\_new\_tools\_attitude'

]

}

}

def select\_next\_question(self, current\_responses, question\_history):

"""

Dynamically select the next question based on previous responses

"""

# Always start with core questions

core\_remaining = [q for q in self.question\_bank['core\_questions']

if q not in current\_responses]

if core\_remaining:

return self.get\_question\_details(core\_remaining[0])

# Check for conditional question triggers

for trigger, questions in self.question\_bank['conditional\_questions'].items():

if self.check\_trigger\_condition(trigger, current\_responses):

remaining = [q for q in questions if q not in current\_responses]

if remaining:

return self.get\_question\_details(remaining[0])

# Check for clarification needs

for situation, questions in self.question\_bank['clarification\_questions'].items():

if self.needs\_clarification(situation, current\_responses):

remaining = [q for q in questions if q not in current\_responses]

if remaining:

return self.get\_question\_details(remaining[0])

# No more questions needed

return None

def check\_trigger\_condition(self, trigger, responses):

"""Check if conditional questions should be triggered"""

if trigger == 'high\_stress\_detected':

stress\_level = responses.get('financial\_stress\_frequency')

return stress\_level in ['Daily', 'Several times a week']

elif trigger == 'entrepreneur\_detected':

employment = responses.get('employment\_status')

return employment == 'Entrepreneur/Business owner'

elif trigger == 'high\_tech\_comfort':

tech\_comfort = int(responses.get('data\_sharing\_comfort', 5))

return tech\_comfort >= 8

return False

#### \*\*Real-time Score Updates\*\*

```python

class RealTimeScreeningScorer:

def \_\_init\_\_(self):

self.min\_questions\_for\_score = 3

self.confidence\_thresholds = {

'low': 0.6,

'medium': 0.75,

'high': 0.9

}

def calculate\_progressive\_score(self, partial\_responses):

"""

Calculate score and confidence as user progresses through questions

"""

num\_responses = len(partial\_responses)

if num\_responses < self.min\_questions\_for\_score:

return {

'score': None,

'confidence': 'insufficient\_data',

'recommendation': 'continue\_questionnaire'

}

# Calculate preliminary score with available data

scorer = MingusScreeningScorer()

partial\_results = scorer.calculate\_success\_probability(partial\_responses)

# Adjust confidence based on how many questions answered

total\_core\_questions = len(scorer.question\_bank['core\_questions'])

completion\_ratio = num\_responses / total\_core\_questions

base\_confidence = partial\_results['overall\_probability']

adjusted\_confidence = base\_confidence \* min(1.0, completion\_ratio \* 1.2)

confidence\_level = self.get\_confidence\_level(adjusted\_confidence, completion\_ratio)

return {

'score': partial\_results['overall\_probability'],

'confidence': confidence\_level,

'recommendation': self.get\_progressive\_recommendation(

partial\_results, confidence\_level

),

'suggested\_next\_questions': self.suggest\_high\_value\_questions(

partial\_responses, partial\_results

)

}

def suggest\_high\_value\_questions(self, current\_responses, current\_results):

"""

Suggest questions that would most improve prediction confidence

"""

missing\_questions = []

# High-impact questions for score refinement

high\_impact\_questions = [

'stress\_management\_interest',

'budgeting\_experience',

'health\_wellness\_priority'

]

for question in high\_impact\_questions:

if question not in current\_responses:

missing\_questions.append({

'question\_id': question,

'impact\_level': 'high',

'reason': 'Critical for Mingus value alignment assessment'

})

return missing\_questions[:2] # Return top 2 suggestions

### \*\*Step 13: Integration with User Journey\*\*

#### \*\*Post-Screening Nurture Sequences\*\*

```python

class ScreeningNurtureManager:

def \_\_init\_\_(self):

self.nurture\_sequences = {

'high\_success\_immediate': {

'trigger': 'score >= 0.75',

'sequence': [

{'delay': 0, 'action': 'send\_welcome\_email', 'template': 'high\_success\_welcome'},

{'delay': 2, 'action': 'send\_onboarding\_reminder', 'template': 'ready\_to\_start'},

{'delay': 7, 'action': 'send\_success\_stories', 'template': 'user\_testimonials'},

{'delay': 14, 'action': 'send\_limited\_time\_offer', 'template': 'early\_adopter\_bonus'}

]

},

'medium\_success\_nurture': {

'trigger': '0.60 <= score < 0.75',

'sequence': [

{'delay': 0, 'action': 'send\_educational\_email', 'template': 'financial\_wellness\_intro'},

{'delay': 3, 'action': 'send\_feature\_highlight', 'template': 'health\_money\_connection'},

{'delay': 7, 'action': 'send\_case\_study', 'template': 'similar\_user\_success'},

{'delay': 14, 'action': 'send\_trial\_invitation', 'template': 'personalized\_trial\_offer'},

{'delay': 21, 'action': 'send\_comparison', 'template': 'mingus\_vs\_alternatives'}

]

},

'educational\_sequence': {

'trigger': '0.45 <= score < 0.60',

'sequence': [

{'delay': 0, 'action': 'send\_resource\_package', 'template': 'financial\_wellness\_guide'},

{'delay': 7, 'action': 'send\_mini\_course', 'template': 'money\_stress\_course\_week1'},

{'delay': 14, 'action': 'send\_mini\_course', 'template': 'money\_stress\_course\_week2'},

{'delay': 21, 'action': 'send\_progress\_check', 'template': 'how\_are\_you\_doing'},

{'delay': 30, 'action': 'send\_re\_screening', 'template': 'ready\_to\_try\_again'}

]

}

}

def enroll\_in\_nurture\_sequence(self, screening\_results, user\_email):

"""

Enroll user in appropriate nurture sequence based on screening results

"""

score = screening\_results['overall\_probability']

# Determine appropriate sequence

sequence\_key = None

for seq\_name, seq\_config in self.nurture\_sequences.items():

if self.evaluate\_trigger(seq\_config['trigger'], score):

sequence\_key = seq\_name

break

if sequence\_key:

# Schedule all emails in the sequence

sequence = self.nurture\_sequences[sequence\_key]['sequence']

for step in sequence:

schedule\_email(

recipient=user\_email,

template=step['template'],

delay\_days=step['delay'],

sequence\_name=sequence\_key,

personalization\_data=screening\_results['personalization\_flags']

)

# Track enrollment

NurtureEnrollment.objects.create(

email=user\_email,

sequence\_name=sequence\_key,

screening\_score=score,

enrollment\_date=timezone.now()

)

def evaluate\_trigger(self, trigger\_expression, score):

"""

Evaluate trigger condition for nurture sequence

"""

# Simple expression evaluator for score-based triggers

if '>=' in trigger\_expression:

threshold = float(trigger\_expression.split('>=')[1].strip())

return score >= threshold

elif '<=' in trigger\_expression and '>=' in trigger\_expression:

# Range evaluation like "0.60 <= score < 0.75"

parts = trigger\_expression.replace('score', str(score))

return eval(parts) # Note: In production, use a safer expression evaluator

return False

## Implementation Timeline & Resource Requirements

### \*\*Phase 1 (Weeks 1-2): Foundation\*\*

- \*\*Week 1\*\*: Data analysis framework, success metrics definition, initial question design

- \*\*Week 2\*\*: Basic scoring algorithm, database schema, preliminary testing

\*\*Resources Needed:\*\*

- 1 Data Analyst (40 hours)

- 1 Backend Developer (30 hours)

- 1 UX Designer (20 hours)

### \*\*Phase 2 (Weeks 3-4): Core Implementation\*\*

- \*\*Week 3\*\*: Frontend questionnaire, API endpoints, basic analytics

- \*\*Week 4\*\*: Integration testing, initial scoring validation, A/B test setup

\*\*Resources Needed:\*\*

- 1 Frontend Developer (40 hours)

- 1 Backend Developer (35 hours)

- 1 QA Engineer (20 hours)

### \*\*Phase 3 (Weeks 5-6): Optimization & Launch\*\*

- \*\*Week 5\*\*: User testing, score calibration, marketing integration

- \*\*Week 6\*\*: Launch preparation, monitoring setup, documentation

\*\*Resources Needed:\*\*

- 1 Marketing Manager (25 hours)

- 1 Data Analyst (30 hours)

- Team coordination (15 hours)

### \*\*Ongoing (Monthly): Optimization\*\*

- Monthly model review and optimization

- Quarterly comprehensive analysis and improvements

\*\*Budget Estimation:\*\*

- \*\*Development\*\*: $8,000-12,000

- \*\*Analytics Tools\*\*: $500-1,000/month

- \*\*A/B Testing Platform\*\*: $200-500/month

- \*\*Email Marketing Integration\*\*: $100-300/month

This screening tool will significantly improve user targeting, reduce churn, and provide valuable insights for product development and marketing optimization.