**Mingus Landing Page Development Guide - Cursor Prompts (CORRECTED VERSION)**

**Execution Order and Dependencies**

Execute these prompts in the exact order listed. Each prompt builds on the previous one's output.

**Prompt 0A: Tax Calculator Implementation (CRITICAL MISSING COMPONENT)**

**Dependencies:** Existing Mingus database schema  
**Estimated Time:** 60-75 minutes  
**Files Created:** backend/services/billing\_features.py, backend/services/tax\_service.py, backend/models/tax\_calculation.py

Create the complete Tax Bill Impact Calculator system that's missing from the current implementation:

Files to create:

- backend/services/billing\_features.py

- backend/services/tax\_service.py

- backend/models/tax\_calculation.py

Implement the exact logic from the MINGUS Calculator Analysis Summary:

1. Tax Calculation Method:

```python

def calculate\_tax(self, customer\_id: int, amount: float, currency: str = 'USD', tax\_exempt: str = None) -> Dict[str, Any]:

# Check if customer is tax exempt

if tax\_exempt == 'exempt' or customer.tax\_exempt == 'exempt':

return {

'tax\_amount': 0.0,

'tax\_rate': 0.0,

'tax\_exempt': True,

'tax\_details': {}

}

# Get customer location for tax calculation

customer\_location = self.\_get\_customer\_location(customer)

# Calculate tax using tax service

tax\_result = self.\_call\_tax\_service(

amount=amount,

currency=currency,

customer\_location=customer\_location,

tax\_exempt=tax\_exempt or customer.tax\_exempt

)

return tax\_result

1. State Tax Rate Configuration:

TAX\_RATES = {

'US': {

'CA': 0.085, 'NY': 0.08, 'TX': 0.0625, 'FL': 0.06,

'WA': 0.065, 'IL': 0.0625, 'PA': 0.06, 'OH': 0.0575,

'default': 0.05

}

}

1. Invoice Amount Calculation:

def \_calculate\_invoice\_amount(self, subscription: Subscription) -> float:

base\_amount = subscription.amount

# Add usage-based charges

usage\_charges = self.\_calculate\_usage\_charges(subscription)

# Calculate tax

tax\_amount = self.\_calculate\_tax\_amount(subscription, base\_amount + usage\_charges)

total\_amount = base\_amount + usage\_charges + tax\_amount

return round(total\_amount, 2)

1. Simple Tax Calculation Fallback:

def \_calculate\_simple\_tax(self, amount: float, customer\_location: Dict[str, str]) -> Dict[str, Any]:

country = customer\_location.get('country', 'US')

state = customer\_location.get('state', '')

if country == 'US':

state\_tax\_rates = {

'CA': 0.085, 'NY': 0.08, 'TX': 0.0625, 'FL': 0.06,

'WA': 0.065, 'IL': 0.0625, 'PA': 0.06, 'OH': 0.0575

}

tax\_rate = state\_tax\_rates.get(state, 0.05) # Default 5%

tax\_amount = amount \* tax\_rate

return {

'tax\_amount': round(tax\_amount, 2),

'tax\_rate': tax\_rate,

'tax\_exempt': False,

'tax\_details': {

'state': state,

'country': country,

'calculation\_method': 'simple'

}

}

Integration requirements:

* Connect to existing Customer and Subscription models
* Use existing database session management patterns
* Include proper error handling and logging
* Support for tax-exempt organizations
* Rounding to 2 decimal places consistently
* Follow existing service class structure and naming conventions

---

## Prompt 0B: Exact Algorithm Implementation (CRITICAL CORRECTIONS)

\*\*Dependencies:\*\* Existing calculator files located

\*\*Estimated Time:\*\* 90-120 minutes

\*\*Files Modified:\*\* `backend/ml/models/intelligent\_job\_matcher.py`, `backend/ml/models/income\_comparator\_optimized.py`, `MINGUS Marketing/src/api/assessmentService.ts`

Replace generic scoring algorithms with the exact formulas from the MINGUS Calculator Analysis Summary:

1. Update backend/ml/models/intelligent\_job\_matcher.py with EXACT formulas:

Multi-Dimensional Job Scoring System:

overall\_score = (

salary\_score \* 0.35 + # 35% weight - Primary importance

skills\_score \* 0.25 + # 25% weight - Skills alignment

career\_score \* 0.20 + # 20% weight - Career progression

company\_score \* 0.10 + # 10% weight - Company quality

location\_score \* 0.05 + # 5% weight - Location fit

growth\_score \* 0.05 # 5% weight - Industry alignment

)

Salary Improvement Score with EXACT thresholds:

def \_calculate\_salary\_improvement\_score(self, job: JobPosting, search\_params: SearchParameters) -> float:

# Calculate percentage increase

salary\_increase = (job.salary\_range.midpoint - search\_params.current\_salary) / search\_params.current\_salary

# Score based on increase percentage - EXACT thresholds

if salary\_increase >= 0.45: # 45%+ increase

return 1.0

elif salary\_increase >= 0.35: # 35%+ increase

return 0.9

elif salary\_increase >= 0.25: # 25%+ increase

return 0.8

elif salary\_increase >= 0.15: # 15%+ increase

return 0.7

elif salary\_increase >= 0.10: # 10%+ increase

return 0.6

elif salary\_increase >= 0.05: # 5%+ increase

return 0.5

else:

return 0.3 # Below 5% increase

Field-Specific Salary Multipliers:

field\_salary\_multipliers = {

FieldType.SOFTWARE\_DEVELOPMENT: 1.2, # 20% premium

FieldType.DATA\_ANALYSIS: 1.1, # 10% premium

FieldType.PROJECT\_MANAGEMENT: 1.0, # Base level

FieldType.MARKETING: 0.95, # 5% discount

FieldType.FINANCE: 1.05, # 5% premium

FieldType.SALES: 0.9, # 10% discount

FieldType.OPERATIONS: 0.95, # 5% discount

FieldType.HR: 0.9 # 10% discount

}

1. Update backend/ml/models/income\_comparator\_optimized.py with EXACT percentile calculation:

@lru\_cache(maxsize=1000)

def \_calculate\_percentile\_cached(self, user\_income: int, median: int, mean: int, p25: int, p75: int) -> float:

# Use simplified normal approximation for speed - EXACT formula

if user\_income <= p25:

return 25.0 \* (user\_income / p25)

elif user\_income <= median:

return 25.0 + 25.0 \* ((user\_income - p25) / (median - p25))

elif user\_income <= p75:

return 50.0 + 25.0 \* ((user\_income - median) / (p75 - median))

else:

# For high incomes, use log-normal approximation

return min(99.9, 75.0 + 24.9 \* (1 - math.exp(-(user\_income - p75) / (mean \* 0.5))))

Career Opportunity Score:

def \_calculate\_career\_opportunity\_score(self, comparisons: List[IncomeComparison]) -> float:

total\_opportunity = 0

total\_weight = 0

for comp in comparisons:

if comp.income\_gap < 0: # Negative gap = opportunity

opportunity = abs(comp.income\_gap) / comp.median\_income \* 100

weight = comp.confidence\_level

total\_opportunity += opportunity \* weight

total\_weight += weight

return total\_opportunity / total\_weight if total\_weight > 0 else 0.0

1. Update MINGUS Marketing/src/api/assessmentService.ts with EXACT point assignments:

private calculateScore(answers: Record<string, any>): { score: number; segment: UserSegment; productTier: ProductTier } {

let totalScore = 0

// EXACT point assignments from Calculator Analysis Summary

const relationshipStatusPoints = {

'single': 0,

'dating': 2,

'serious\_relationship': 4,

'married': 6,

'complicated': 8

}

const spendingHabitsPoints = {

'keep\_separate': 0,

'share\_some': 2,

'joint\_accounts': 4,

'spend\_more\_relationships': 6,

'overspend\_impress': 8

}

const financialStressPoints = {

'never': 0,

'rarely': 2,

'sometimes': 4,

'often': 6,

'always': 8

}

const emotionalTriggersPoints = {

'after\_breakup': 3,

'after\_arguments': 3,

'when\_lonely': 2,

'when\_jealous': 2,

'social\_pressure': 2,

'none': 0

}

// Calculate score based on answers using EXACT point system

Object.entries(answers).forEach(([questionId, answer]) => {

// Implementation continues with exact scoring...

})

// EXACT segment classification

let segment: UserSegment

let productTier: ProductTier

if (totalScore <= 16) {

segment = 'stress-free'

productTier = 'Budget ($10)'

} else if (totalScore <= 25) {

segment = 'relationship-spender'

productTier = 'Mid-tier ($20)'

} else if (totalScore <= 35) {

segment = 'emotional-manager'

productTier = 'Mid-tier ($20)'

} else {

segment = 'crisis-mode'

productTier = 'Professional ($50)'

}

return { score: totalScore, segment, productTier }

}

Performance Requirements:

* Income comparator must achieve <500ms calculation time (target 45ms average from documented achievement)
* LRU cache with maxsize=1000 for percentile calculations
* Memory-efficient immutable data structures
* Thread-safe operations with proper locking

---

## Prompt 0C: Architecture Integration (CRITICAL CONNECTIONS)

\*\*Dependencies:\*\* Prompts 0A-0B completed

\*\*Estimated Time:\*\* 45-60 minutes

\*\*Files Modified:\*\* Integration points across existing system

Connect calculator systems with existing MINGUS project file structure:

1. Integration Points with Existing Files:
   * Import calculation patterns from backend/ml/models/intelligent\_job\_matcher.py
   * Use performance optimizations from backend/ml/models/income\_comparator\_optimized.py
   * Connect to existing MINGUS Marketing/src/api/assessmentService.ts structure
   * Integrate with backend/services/billing\_features.py for tax calculations
2. Database Integration:
   * Connect to existing user profile system with 25+ fields (users, user\_profiles tables)
   * Use existing subscription management with 3 tiers: $10, $20, $50 (subscription\_plans table)
   * Integrate with existing authentication patterns and decorators
   * Connect to existing customer and billing models
3. Performance Requirements (from documented achievements):
   * Maintain 45ms average calculation time for income comparisons
   * Use existing LRU caching patterns: @lru\_cache(maxsize=1000)
   * Implement PerformanceMonitor class with sub-500ms targets
   * Memory-efficient data structures using frozen dataclasses
4. Cultural Personalization for Target Demographic:
   * African American professionals, ages 25-35, income $40K-$100K
   * Target metro areas: Atlanta (+95,000), Houston (+88,000), DC Metro (+75,000), Dallas-Fort Worth (+72,000), NYC (+65,000), Philadelphia (+58,000), Chicago (+52,000), Charlotte (+48,000), Miami (+42,000), Baltimore (+35,000)
   * Community-specific challenges: Income instability, student debt, career path barriers, homeownership challenges, financial literacy gaps
   * Age-based personalization: 25-35 focus on career advancement, student loan management, home ownership goals
5. Data Sources Integration:
   * 2022 American Community Survey (ACS) data for demographic comparisons
   * Bureau of Labor Statistics data for income benchmarking
   * Real salary data from LinkedIn, Indeed, Glassdoor, ZipRecruiter
   * Confidence intervals and sample sizes for statistical reliability
6. Service Integration Patterns:
   * Follow existing service class structure and naming conventions
   * Use existing database connection and session management
   * Implement existing error handling and logging patterns
   * Connect to existing monitoring and analytics infrastructure

Ensure all integrations maintain compatibility with existing codebase architecture while implementing the exact calculation specifications from the MINGUS Calculator Analysis Summary.

---

## Prompt 1: Database Schema Setup

\*\*Dependencies:\*\* Prompts 0A-0C completed

\*\*Estimated Time:\*\* 30-45 minutes

\*\*Files Created:\*\* `migrations/add\_assessment\_tables.py`

Create a database migration file for the Mingus application that adds assessment functionality to the existing PostgreSQL schema. The migration should:

1. Create an `assessments` table with:

- id (UUID primary key)

- type (enum: 'ai\_job\_risk', 'relationship\_impact', 'tax\_impact', 'income\_comparison')

- title (string)

- questions\_json (JSONB - stores assessment questions)

- scoring\_config (JSONB - stores scoring logic from exact formulas)

- active (boolean, default true)

- created\_at, updated\_at timestamps

2. Create a `user\_assessments` table with:

- id (UUID primary key)

- user\_id (UUID, foreign key to existing users table, nullable for anonymous users)

- assessment\_id (UUID, foreign key to assessments table)

- responses\_json (JSONB - stores user answers)

- score (integer)

- risk\_level (string)

- completed\_at (timestamp)

- email (string - for anonymous users)

- first\_name (string - for anonymous users)

- location (string - optional)

- job\_title (string - optional)

- industry (string - optional)

3. Create an `assessment\_results` table with:

- id (UUID primary key)

- user\_assessment\_id (UUID, foreign key)

- insights\_json (JSONB - personalized insights)

- recommendations\_json (JSONB - action items)

- automation\_score (integer - for AI assessment)

- augmentation\_score (integer - for AI assessment)

- cost\_projections (JSONB - financial impact calculations)

- created\_at timestamp

4. Create indexes for performance:

- user\_assessments.user\_id

- user\_assessments.assessment\_id

- user\_assessments.email

- assessment\_results.user\_assessment\_id

Follow the existing Mingus database patterns and naming conventions. Include proper foreign key constraints, data validation, and ensure compatibility with existing SQLAlchemy models.

---

## Prompt 2: Assessment Data Seeding with Exact Specifications

\*\*Dependencies:\*\* Prompts 0A-0C and 1 completed

\*\*Estimated Time:\*\* 45-60 minutes

\*\*Files Created:\*\* `scripts/seed\_assessments.py`

Create a data seeding script that populates the assessments table with the EXACT assessment configurations from the MINGUS Calculator Analysis Summary for the 4 lead magnets:

1. AI Job Risk Calculator - EXACT SPECIFICATIONS:

- Use the precise job risk database with 700+ occupations and specific automation/augmentation percentages

- Industry modifiers: Technology (+10% automation), Finance (+5%), Healthcare (-10%), Education (-15%)

- Experience adjustments: 10+ years gets -5% automation risk

- AI usage bonus: Daily users get -10% automation risk, +15% augmentation

- Technical skills: High/expert gets -8% automation, +12% augmentation

- Final risk calculation: automation\_score \* 0.7 + augmentation\_score \* 0.3

2. Relationship Impact Calculator - EXACT POINT SYSTEM:

- Relationship status: Single (0), Dating (2), Serious (4), Married (6), Complicated (8)

- Spending habits: Separate finances (0), Share some (2), Joint accounts (4), Spend more in relationships (6), Overspend to impress (8)

- Financial stress: Never (0), Rarely (2), Sometimes (4), Often (6), Always (8)

- Emotional triggers (checkbox): After breakup (3), After arguments (3), When lonely (2), When jealous (2), Social pressure (2)

- EXACT segments: Stress-Free (0-16), Relationship-Spender (17-25), Emotional-Manager (26-35), Crisis-Mode (36+)

3. Tax Impact Calculator - EXACT TAX LOGIC:

- State tax rates: CA (8.5%), NY (8%), TX (6.25%), FL (6%), WA (6.5%), IL (6.25%), PA (6%), OH (5.75%), default (5%)

- Cost projections: Tax inefficiency $1,200/year + parent care costs $2,400/year + benefit losses $1,800/year

- 2025 tax policy impact calculations based on actual tax law changes

4. Income Comparison Calculator - EXACT DEMOGRAPHIC GROUPS:

- National Median, African American, Age Group (25-35), African American Ages 25-35

- College Graduates, African American College Graduates, Metro Area, African American Metro

- Target metros: Atlanta, Houston, DC, Dallas, NYC, Philadelphia, Chicago, Charlotte, Miami, Baltimore

- BLS data integration with confidence intervals and sample sizes

Include the exact calculation formulas, not generic approximations. Reference the existing file locations where applicable.

Each assessment should include:

- Realistic question text and answer options

- Proper scoring algorithms with exact mathematical formulas

- Meaningful output calculations with specific dollar amounts

- Personalized recommendations based on precise score ranges

- No fake data - use actual research and realistic projections

---

## Prompt 3: Backend API Endpoints

\*\*Dependencies:\*\* Prompts 1-2 completed

\*\*Estimated Time:\*\* 60-90 minutes

\*\*Files Created:\*\* `backend/routes/assessment\_routes.py`, `backend/services/assessment\_service.py`

Create Flask API endpoints for the Mingus assessment system that integrate with the existing backend architecture and exact calculation implementations:

1. GET /api/assessments/available

- Returns list of active assessments with metadata

- Include real completion counts from user\_assessments table

- Calculate average completion time from database

- Integrate with existing authentication patterns (check if user is logged in)

- Return different data for authenticated vs anonymous users

2. POST /api/assessments/{type}/submit

- Accept assessment responses (anonymous or authenticated users)

- Validate responses against assessment schema from database

- Calculate scores using the EXACT seeded scoring logic (from Prompts 0A-0C)

- Store results in user\_assessments and assessment\_results tables

- For anonymous users: create lead record with email/first\_name

- Return immediate results (free tier insights) + conversion offer

- Integrate with existing user creation patterns

- Include lead scoring for sales prioritization

3. GET /api/assessments/{user\_assessment\_id}/results

- Return detailed assessment results for specific assessment

- Include personalized insights and recommendations based on exact calculations

- Check user authorization (own results only)

- Integrate with existing subscription system for premium features

- Return different detail levels based on subscription tier

4. POST /api/assessments/convert/{user\_assessment\_id}

- Handle conversion from free assessment to paid subscription

- Integrate with existing Stripe payment processing

- Update user profile with assessment insights

- Trigger email sequences based on assessment type and score

- Track conversion attribution

5. GET /api/assessments/stats

- Return real-time statistics for social proof

- Total assessments completed today/this week

- Average scores by assessment type

- Anonymous aggregated data only

Implementation requirements:

- Use existing authentication decorators (@require\_auth, @optional\_auth)

- Follow existing error handling patterns with proper HTTP status codes

- Integrate with existing database session management

- Include proper input validation and sanitization

- Use existing logging and monitoring patterns

- Include rate limiting for anonymous users

- Proper CORS handling for frontend integration

- Connect to exact calculation services from Prompts 0A-0C

---

## Prompt 4: Assessment Scoring Service with Exact Formulas

\*\*Dependencies:\*\* Prompts 1-3 completed

\*\*Estimated Time:\*\* 45-60 minutes

\*\*Files Created/Modified:\*\* `backend/services/assessment\_calculator.py`, integrate with existing `backend/ml/models/` files

Create the assessment scoring service that implements the EXACT calculation logic from the MINGUS Calculator Analysis Summary:

1. AI Job Risk Calculator - IMPLEMENT EXACT ALGORITHM:

```python

# Base scoring from existing intelligent\_job\_matcher.py patterns

overall\_score = (

salary\_score \* 0.35 + # 35% weight - Primary importance

skills\_score \* 0.25 + # 25% weight - Skills alignment

career\_score \* 0.20 + # 20% weight - Career progression

company\_score \* 0.10 + # 10% weight - Company quality

location\_score \* 0.05 + # 5% weight - Location fit

growth\_score \* 0.05 # 5% weight - Industry alignment

)

# Use existing field\_salary\_multipliers from intelligent\_job\_matcher.py

# Final risk level: automation\_score \* 0.7 + augmentation\_score \* 0.3

1. Relationship Impact Calculator - EXACT POINT SYSTEM:

# Implement exact scoring from assessmentService.ts

def calculate\_relationship\_score(responses):

total\_score = 0

# Relationship status points

relationship\_points = {

'single': 0, 'dating': 2, 'serious': 4,

'married': 6, 'complicated': 8

}

# Financial stress frequency

stress\_points = {

'never': 0, 'rarely': 2, 'sometimes': 4,

'often': 6, 'always': 8

}

# Emotional spending triggers (additive)

trigger\_points = {

'after\_breakup': 3, 'after\_arguments': 3,

'when\_lonely': 2, 'when\_jealous': 2, 'social\_pressure': 2

}

# EXACT segment classification

if total\_score <= 16: return 'stress-free', 'Budget ($10)'

elif total\_score <= 25: return 'relationship-spender', 'Mid-tier ($20)'

elif total\_score <= 35: return 'emotional-manager', 'Mid-tier ($20)'

else: return 'crisis-mode', 'Professional ($50)'

1. Tax Impact Calculator - INTEGRATE WITH billing\_features.py:

# Use TAX\_RATES from billing\_features.py

# State tax considerations for target metro areas

# 2025 tax policy impact assessment based on income level and family situation

# Cost projections: Tax inefficiency $1,200 + parent care $2,400 + benefit losses $1,800

1. Income Comparison Calculator - EXACT PERCENTILE FORMULA:

# Integrate with existing income\_comparator\_optimized.py

# Use the exact \_calculate\_percentile\_cached method

# 8 demographic comparison groups from the Calculator Analysis

# Performance target: <500ms calculation (45ms average achieved)

Performance Requirements:

* Achieve 45ms average calculation time for income comparisons
* Use LRU caching with maxsize=1000 from existing code
* Memory-efficient immutable data structures
* Thread-safe operations with proper locking

Integration Requirements:

* Import from existing files: intelligent\_job\_matcher.py, income\_comparator\_optimized.py, assessmentService.ts
* Use existing database patterns and error handling
* Connect to billing\_features.py for tax calculations

Each calculator should:

* Return detailed breakdown of scoring logic
* Provide specific, actionable insights
* Include personalized recommendations based on individual responses
* Calculate realistic financial impact projections using exact formulas
* Generate segment/risk classifications for targeted follow-up
* Include confidence intervals for projections where appropriate

Use existing Mingus patterns for:

* Database queries and connections
* Error handling and logging
* Data validation and sanitization
* Service class structure and naming

**Prompt 5: React Assessment Components**

**Dependencies:** Prompts 1-4 completed  
**Estimated Time:** 90-120 minutes  
**Files Created:** Multiple component files in frontend/src/components/assessments/

Create React assessment components for the Mingus application that match the existing UI patterns and design system:

1. AssessmentLanding.tsx:
   * Responsive grid (4 assessments on desktop, 2x2 on tablet, 1 column on mobile)
   * Real completion statistics from /api/assessments/available endpoint
   * Clear value propositions highlighting specific outcomes for each assessment
   * Trending/popular badges based on actual usage data
   * Estimated completion times from real user data
   * Matches existing Mingus color scheme: primary purple (#8A31FF), green (#10b981)
   * Uses existing typography (Inter, Open Sans fonts)
2. AssessmentFlow.tsx:
   * Multi-step form with visual progress tracking
   * Question types: text input, select dropdown, multi-select checkboxes, radio button groups
   * Real-time validation using existing form validation patterns
   * Auto-save progress to localStorage with recovery on page reload
   * Mobile-optimized with 44px minimum touch targets
   * Smooth transitions between steps
   * Back/Next navigation with validation
   * Loading states during submission
3. AssessmentResults.tsx:
   * Risk level display with color-coded indicators (red/yellow/green)
   * Score visualization using charts/progress bars
   * Personalized insights section with specific recommendations from exact calculations
   * Cost projection display with dollar amounts and time frames
   * Social comparison ("You scored higher than 65% of users")
   * Conversion offer modal with real pricing from database
   * PDF download functionality for results
   * Social sharing buttons for assessment completion
4. ConversionModal.tsx:
   * Integration with existing subscription tiers ($10, $20, $50)
   * Real-time countdown timer (60 minutes, server-validated)
   * Stripe payment integration using existing payment components
   * User testimonials from actual users (pull from database)
   * Exit-intent detection with emergency $19 offer
   * Mobile-optimized checkout flow
5. AssessmentQuestion.tsx:
   * Reusable question component handling all input types
   * Progress indicator showing current question / total questions
   * Skip logic for conditional questions
   * Input validation with helpful error messages
   * Accessibility features (proper labels, keyboard navigation, screen reader support)

Design Requirements:

* Follow existing Mingus design system colors, spacing, and typography
* Responsive design working on 320px to 1920px+ screens
* Loading states and skeleton screens for better perceived performance
* Error boundaries with user-friendly error messages
* Proper form validation with real-time feedback
* Consistent with existing app navigation and interaction patterns
* WCAG 2.1 AA accessibility compliance

Technical Requirements:

* Use existing state management patterns (Redux/Context)
* Integrate with existing authentication system
* API calls using existing HTTP client configuration
* Error handling consistent with existing error handling patterns
* TypeScript interfaces for all props and state
* Unit tests for all components
* Integration with exact calculation results from backend services

**Prompt 6: Responsive Landing Page**

**Dependencies:** Prompts 1-5 completed  
**Estimated Time:** 120-150 minutes  
**Files Created:** frontend/public/landing.html, associated CSS and JS files

Create a responsive landing page that addresses all identified weaknesses from the analysis:

DESIGN REQUIREMENTS:

1. Responsive design scaling from 320px (mobile) to 1920px+ (desktop)
2. Matches Mingus application aesthetic:
   * Consistent spacing and border radius with existing app
   * Professional, clean design appropriate for target demographic
3. Real data integration:
   * Social proof counters query /api/assessments/stats endpoint
   * Assessment completion rates from actual database
   * User testimonials from verified users
   * No fake metrics or false promises

FUNCTIONAL REQUIREMENTS:

1. Working Assessment Integration:
   * Assessment selection opens functional assessment flow with exact calculations
   * Real assessment submission and results using implemented scoring algorithms
   * Lead capture with database integration
   * Email collection for anonymous users
2. Interactive Elements:
   * FAQ section with expand/collapse functionality
   * Progressive disclosure of information
   * Smooth animations and transitions
   * Proper loading states
3. Conversion Optimization:
   * Clear value propositions for each assessment
   * Specific outcome promises (dollar amounts, time savings)
   * Social proof from real users
   * Urgency elements with backend validation

CONTENT STRUCTURE:

* Hero section with main value proposition and primary CTA
* Assessment selection grid with 4 working options (including Tax Impact Calculator)
* Social proof section with real completion statistics
* Feature highlights with specific benefits
* User testimonials from actual database
* FAQ section addressing common concerns
* Final conversion CTA with clear next steps

PERFORMANCE REQUIREMENTS:

* Page load time under 3 seconds on 3G connection
* Lighthouse score of 90+ for Performance, Accessibility, Best Practices, SEO
* Progressive enhancement (works without JavaScript)
* Proper meta tags and Open Graph data
* Mobile-first CSS with desktop enhancements

ANALYTICS INTEGRATION:

* Google Analytics 4 event tracking for all interactions
* Microsoft Clarity tracking
* Conversion tracking for assessment completions
* Lead generation tracking
* User journey mapping
* A/B testing framework ready

ACCESSIBILITY REQUIREMENTS:

* WCAG 2.1 AA compliance
* Proper heading hierarchy and semantic HTML
* Alt text for all images
* Keyboard navigation support
* Screen reader compatibility
* Color contrast ratios meeting standards

Include proper error handling, loading states, and offline functionality where appropriate.

**Prompt 7: Analytics and Tracking Implementation**

**Dependencies:** Prompts 1-6 completed  
**Estimated Time:** 45-60 minutes  
**Files Created:** frontend/src/utils/analytics.js, backend/analytics/assessment\_tracking.py

Create comprehensive analytics tracking for the new landing page and assessment system:

1. Frontend Event Tracking:
   * assessment\_landing\_viewed (track which assessments users view)
   * assessment\_started (with assessment\_type parameter)
   * assessment\_question\_answered (track completion funnel)
   * assessment\_completed (with score, risk\_level, conversion\_eligible)
   * email\_captured (lead generation event with source tracking)
   * conversion\_modal\_opened (track conversion interest)
   * payment\_initiated (track conversion attempts)
   * social\_proof\_interaction (track credibility element engagement)
2. Backend Analytics Endpoints:
   * POST /api/analytics/track-event (central event tracking)
   * GET /api/analytics/dashboard (real-time metrics for admin)
   * GET /api/analytics/conversion-funnel (assessment completion rates)
   * GET /api/analytics/lead-quality (scoring and segmentation metrics)
3. Conversion Funnel Analysis:
   * Landing page views → Assessment starts → Completions → Lead capture → Paid conversion
   * Drop-off analysis at each step
   * Time-to-completion metrics
   * Device and browser performance analysis
4. Real-time Dashboard Metrics:
   * Live assessment completion counter for social proof
   * Conversion rates by assessment type
   * Revenue attribution by lead source
   * User engagement heatmaps
   * Geographic distribution of users
5. Performance Monitoring:
   * Page load times by device type
   * Assessment completion rates
   * API response times
   * Error rates and types
   * Database query performance

Integration Requirements:

* Use existing Google Analytics 4 configuration
* Connect to existing Microsoft Clarity setup
* Integrate with current database logging patterns
* Include proper GDPR compliance and privacy controls
* Real-time updates for social proof counters without fake data
* Attribution tracking from marketing campaigns
* Revenue tracking through existing Stripe integration

Data Privacy Compliance:

* Anonymous user tracking with proper consent management
* PII protection in analytics data
* User opt-out capabilities
* Data retention policies alignment
* GDPR and CCPA compliance measures

**Prompt 8: Testing and Quality Assurance**

**Dependencies:** All previous prompts completed  
**Estimated Time:** 90-120 minutes  
**Files Created:** Multiple test files and CI/CD configurations

Create a comprehensive test suite for the new landing page and assessment system:

1. Backend API Tests:
   * Unit tests for all assessment endpoints (/api/assessments/\*)
   * Assessment scoring algorithm accuracy tests (verify exact formulas from Calculator Analysis)
   * Database integration tests with real data scenarios
   * Authentication and authorization tests
   * Rate limiting and security tests
   * Performance tests for database queries
2. Frontend Component Tests:
   * React component unit tests using Jest and React Testing Library
   * Assessment flow integration tests
   * Form validation tests
   * Responsive design tests (320px to 1920px+)
   * Accessibility tests using jest-axe
   * Cross-browser compatibility (Chrome, Firefox, Safari, Edge)
3. End-to-End Tests:
   * Complete user journey: landing → assessment → results → conversion
   * Anonymous user flow (email capture)
   * Authenticated user flow (profile integration)
   * Payment processing flow
   * Mobile device testing (iOS Safari, Chrome Mobile)
   * Error handling and edge cases
4. Performance Tests:
   * Page load speed optimization (target: <3s on 3G)
   * Assessment submission performance
   * Database query optimization (verify 45ms income comparison target)
   * Concurrent user load testing
   * Memory leak detection
   * Mobile performance benchmarks
5. Security Tests:
   * Input validation and SQL injection prevention
   * XSS protection verification
   * CSRF token validation
   * Authentication bypass attempts
   * Rate limiting effectiveness
   * Data privacy compliance verification
6. Analytics Verification:
   * Event tracking accuracy
   * Conversion funnel data integrity
   * Real-time metrics validation
   * Privacy compliance in tracking
   * Revenue attribution accuracy
7. Mathematical Accuracy Tests:
   * Verify exact calculation formulas match Calculator Analysis Summary
   * Test salary improvement score thresholds (45%=1.0, 35%=0.9, etc.)
   * Validate relationship scoring point assignments
   * Confirm percentile calculation accuracy
   * Test tax calculation with different state rates

Test Coverage Requirements:

* Backend: 90%+ code coverage for assessment-related functions
* Frontend: 85%+ coverage for assessment components
* Integration tests for all user-facing workflows
* Performance benchmarks for all critical paths
* Security tests for all input vectors
* Mathematical accuracy tests for all calculation formulas

CI/CD Integration:

* Automated test runs on all pull requests
* Performance regression detection (maintain 45ms income comparison target)
* Accessibility compliance verification
* Security scanning integration
* Database migration testing
* Deployment smoke tests

Quality Gates:

* All tests must pass before deployment
* Performance thresholds must be met (45ms income calculation average)
* Accessibility scores must maintain AA compliance
* Security scans must show no high/critical vulnerabilities
* Database migrations must be reversible
* Mathematical accuracy must match documented specifications

**Implementation Checklist - UPDATED WITH CRITICAL CORRECTIONS**

**Phase 0: Critical Corrections (Prompts 0A-0C)**

* [ ] Tax Bill Impact Calculator implemented in billing\_features.py with exact state tax rates
* [ ] Exact formula implementations added to existing calculator files
* [ ] Salary improvement scoring with precise thresholds (45%=1.0, 35%=0.9, etc.)
* [ ] Relationship assessment point system matches documented specifications
* [ ] Income percentile calculation uses exact mathematical formulas
* [ ] Field-specific salary multipliers integrated from existing code

**Phase 1: Foundation (Prompts 1-2)**

* [ ] Database migration created and tested with assessment tables
* [ ] Assessment data seeded with EXACT point systems and scoring ranges
* [ ] Database indexes created for performance optimization
* [ ] Data validation constraints working with real calculation logic

**Phase 2: Backend (Prompts 3-4)**

* [ ] API endpoints implemented with integration to existing calculator files
* [ ] Assessment scoring algorithms use exact formulas from Analysis Summary
* [ ] Tax calculator endpoints connect to billing\_features.py
* [ ] Performance targets met: 45ms average for income comparisons
* [ ] Integration with existing authentication system verified

**Phase 3: Frontend (Prompt 5)**

* [ ] React components integrate with exact scoring from assessmentService.ts
* [ ] Assessment flow reflects precise point assignments and segments
* [ ] Mobile responsiveness verified across all calculator types
* [ ] Integration with backend APIs returns accurate calculated results

**Phase 4: Landing Page (Prompt 6)**

* [ ] Responsive landing page connects to working calculator implementations
* [ ] Real data integration shows actual calculation results
* [ ] All four calculator types (including Tax Impact) functional
* [ ] Performance optimization meets <3 second load requirement

**Phase 5: Analytics & Testing (Prompts 7-8)**

* [ ] Analytics tracking captures exact calculation results
* [ ] Test suite verifies mathematical accuracy of all formulas
* [ ] Performance benchmarks confirm 45ms income calculation target
* [ ] Security requirements verified for all calculator endpoints

**Notes for Implementation**

1. **Execute prompts sequentially** - each builds on the previous
2. **Critical Phase 0 must be completed first** - addresses fundamental missing components
3. **Test thoroughly at each phase** - don't proceed if current phase has issues
4. **Use real data only** - no fake metrics or false promises
5. **Maintain consistency** with existing Mingus patterns
6. **Mobile-first approach** throughout all development
7. **Security considerations** at every step
8. **Performance monitoring** from day one with documented targets

**Critical Success Criteria - UPDATED**

* **Mathematical Accuracy**: All calculations match the exact formulas in MINGUS Calculator Analysis Summary
* **Performance Targets**: Income comparator achieves 45ms average (documented achievement)
* **Complete Implementation**: All 4 calculators working, including previously missing Tax Impact Calculator
* **Integration Compliance**: Uses existing files (intelligent\_job\_matcher.py, income\_comparator\_optimized.py, assessmentService.ts, billing\_features.py)
* **Exact Scoring**: Point assignments, segment ranges, and risk calculations match documented specifications precisely

**Success Metrics**

* **Conversion Rate**: Target 15-25% from assessment completion to email capture
* **Assessment Completion Rate**: Target 70%+ completion rate
* **Page Performance**: <3 second load time, 90+ Lighthouse score
* **User Experience**: Smooth flow from landing → assessment → results
* **Real Data**: All social proof and metrics from actual database queries
* **Mathematical Precision**: All calculations verified against MINGUS Calculator Analysis Summary