Cursor Prompts for Mingus UX Optimization

Copy and paste these prompts into Cursor in order. Each prompt is designed to complete one specific task.

Priority 0: Landing Page - Remove Duplicate Assessment Buttons

Prompt 1: Add Anchor Navigation to HeroSection

In frontend/src/components/sections/HeroSection.tsx, modify the assessment buttons to use anchor navigation instead of directly opening the assessment modal.

1. Add a new function called `scrollToAssessment` that:
   - Takes an assessmentType parameter
   - Scrolls smoothly to an element with id="assessments"
   - After a 600ms delay, calls onAssessmentClick with the assessmentType

2. Update all 4 assessment buttons (ai-risk, income-comparison, cuffing-season, layoff-risk) to call scrollToAssessment instead of directly calling onAssessmentClick

3. Keep the "Get Started" button unchanged - it should still navigate to /signup

The scroll should use behavior: 'smooth' and block: 'start'.

Prompt 2: Create Unified Assessment Section

In frontend/src/components/LandingPage.tsx, replace the "Risk Assessment Preview" section (the section with className containing "py-8" and aria-label="Career risk assessment preview" around lines 483-519) with a new unified assessment section.

The new section should:
1. Have id="assessments" (this is the anchor target)
2. Have a dark gradient background (from-gray-900 to-gray-800)
3. Include a header: "Discover Your Financial & Career Profile"
4. Include a subheader explaining the free assessments
5. Display 4 assessment cards in a responsive grid (1 col mobile, 2 col tablet, 4 col desktop)

Each card should:
- Be a button that calls setActiveAssessment with the appropriate type
- Show an icon (Shield for ai-risk, TrendingUp for income-comparison, Heart for cuffing-season, Briefcase for layoff-risk)

- Display a "FREE" badge in green
- Show the assessment name and a brief description
- Show estimated time (3-5 min, 2-3 min, 3-4 min, 4-5 min respectively)
- Have hover effects (scale, border color change)

Below the cards, add a text link: "Skip assessments and sign up directly" that navigates to /signup?source=direct

Import the Clock icon from lucide-react for the time estimates.

Prompt 3: Remove AssessmentSection Component

In frontend/src/components/LandingPage.tsx:

1. Remove the import statement for AssessmentSection (import AssessmentSection from './sections/AssessmentSection')

2. Remove the <AssessmentSection /> component usage from the JSX (it should be around lines 521-526, wrapped in a comment "Assessment Section")

3. Delete the file frontend/src/components/sections/AssessmentSection.tsx entirely

The unified assessment section we created in the previous step replaces this component.

Priority 1: New User Process - Remove Friction

Prompt 4: Replace Password Confirmation with Show/Hide Toggle

In the signup page component (check frontend/src/pages/SignUpPage.tsx or wherever the registration form is located):

1. Add state for password visibility: const [showPassword, setShowPassword] = useState(false)

2. Import Eye and EyeOff icons from lucide-react

3. Remove the "Confirm Password" field entirely

4. Modify the Password field to:
   - Use type={showPassword ? "text" : "password"}
   - Add a button inside the input wrapper (position absolute, right side) that toggles showPassword
   - The button should show EyeOff icon when password is visible, Eye icon when hidden
   - Add aria-label for accessibility

- Add helper text below: "Must be at least 8 characters"

5. Remove any validation logic that compares password to confirmPassword

6. Update form validation to only check password length (min 8 characters)

Prompt 5: Create QuickSetupOverlay Component

Create a new file frontend/src/components/QuickSetupOverlay.tsx

This is a modal overlay component with these specifications:

Props interface:
- isOpen: boolean
- onClose: () => void
- onComplete: (data: { incomeRange: string; primaryGoal: string }) => void

Features:
1. Dark semi-transparent backdrop (bg-black/70 with backdrop-blur)
2. Centered modal card (max-w-md, bg-gray-800, rounded-2xl)
3. Close button (X icon) in top right corner that calls onClose
4. Header: "Quick Personalization" with subtext "Just 2 questions to personalize your experience"

Question 1 - Income Range:
- Label with DollarSign icon: "Annual Income Range"
- 4 buttons in 2x2 grid: $30,000-$50,000, $50,000-$75,000, $75,000-$100,000, $100,000+
- Selected state: violet background

Question 2 - Primary Goal:
- Label with Target icon: "Top Financial Priority"
- 5 buttons stacked vertically: Build Emergency Fund, Pay Off Debt, Start Investing, Save for Home, Plan for Retirement
- Selected state: violet background

Actions:
- "Continue to Dashboard" button (disabled until both selected, violet gradient when enabled)
- "I'll do this later" text button that calls onClose

On submit, POST to /api/profile/quick-setup with { incomeRange, primaryGoal }, then call onComplete.

Return null if !isOpen.

Prompt 6: Integrate QuickSetupOverlay into Dashboard

In the main dashboard component (frontend/src/components/CareerDashboard.tsx or similar):

1. Import QuickSetupOverlay from './QuickSetupOverlay'

2. Add state: const [showQuickSetup, setShowQuickSetup] = useState(false)

3. Add a useEffect that runs on mount to check if the user has completed setup:
   - Fetch GET /api/profile/setup-status with credentials: 'include'
   - If response.setupCompleted is false, setShowQuickSetup(true)
   - Wrap in try/catch, fail silently if the endpoint doesn't exist yet

4. Add the QuickSetupOverlay component to the render, just before the closing tag of the main container:

```
<QuickSetupOverlay
  isOpen={showQuickSetup}
  onClose={() => setShowQuickSetup(false)}
  onComplete={() => setShowQuickSetup(false)}
/>
```

Prompt 7: Update Registration to Skip Separate Quick Setup Page

In the signup/registration component and App.tsx routing:

1. In the registration success handler, change the navigation from '/quick-setup' or '/profile-setup' to '/career-dashboard' (or whatever the main dashboard route is)

2. In App.tsx, you can keep the /quick-setup route for now but it's no longer part of the main flow

3. After successful registration, the user should go directly to the dashboard where the QuickSetupOverlay will appear

4. Make sure the assessment data from localStorage (mingus_assessment) is still being used to pre-fill the email and firstName on the signup form

---

Priority 1: Returning User Process

Prompt 8: Add Remember Me to Login Page

In the login page component (check frontend/src/pages/LoginPage.tsx or the login section in App.tsx):

1. Add state: const [rememberMe, setRememberMe] = useState(false)

2. After the password field, add a flex container with justify-between containing:

   Left side - Remember Me checkbox:
   - A label with flex items-center gap-2
   - Checkbox input (styled: w-4 h-4, rounded, border-gray-600, bg-gray-700, text-violet-600)
   - Text: "Remember me for 30 days" (text-sm text-gray-400)

   Right side - Forgot Password link:
   - A button with onClick={() => navigate('/forgot-password')}
   - Text: "Forgot password?" (text-sm text-violet-400 hover:underline)

3. Update the login fetch call to include rememberMe in the request body:
   body: JSON.stringify({ email, password, rememberMe })

Prompt 9: Create Forgot Password Page

Create a new file frontend/src/pages/ForgotPasswordPage.tsx

This page should have:

1. Centered layout with max-w-md container
2. Header: "Reset Your Password"
3. Subtext: "Enter your email and we'll send you a reset link"

4. Form with:
   - Email input field
   - Submit button: "Send Reset Link" (violet gradient)
   - Back to login link

5. States to handle:
   - isSubmitting (show loading state on button)
   - isSuccess (show success message: "Check your email for the reset link")
   - error (show error message)

6. On submit, POST to /api/auth/forgot-password with { email }

7. Add this route to App.tsx: <Route path="/forgot-password" element={<ForgotPasswordPage />} />

Prompt 10: Backend - Update Login Endpoint for Remember Me

In backend/api/auth_endpoints.py (or wherever the login endpoint is defined):

Update the login endpoint to handle the rememberMe parameter:

1. Extract rememberMe from the request JSON (default to False)

2. When creating the JWT token, set the expiry based on rememberMe:
   - If rememberMe is True: expiry = timedelta(days=30)
   - If rememberMe is False: expiry = timedelta(hours=24)

3. Pass this expiry to your create_jwt_token function (or jwt.encode)

4. The response should remain the same - the token will just have a different expiration

---

Prompt 11: Backend - Create Quick Setup Endpoint

In the backend, create or update the quick setup endpoint:

File: backend/api/profile_endpoints.py (or create backend/api/quick_setup_endpoints.py)

Create two endpoints:

1. POST /api/profile/quick-setup
   - Requires authentication (@login_required or JWT verification)
   - Accepts JSON body: { incomeRange: string, primaryGoal: string }
   - Saves to user_profiles table with fields: user_id, income_range, primary_goal, setup_completed=True
   - Returns: { success: true, message: "Profile setup completed" }

2. GET /api/profile/setup-status
   - Requires authentication
   - Queries user_profiles table for current user
   - Returns: { setupCompleted: boolean }

- If no profile exists, return { setupCompleted: false }

Make sure to register the blueprint in your Flask app if creating a new file.

Prompt 12: Backend - Create Forgot Password Endpoint

In backend/api/auth_endpoints.py, add a forgot password endpoint:

POST /api/auth/forgot-password
- Accepts JSON body: { email: string }
- Look up user by email
- If user exists:
  - Generate a password reset token (random string or JWT with short expiry)
  - Save token to database with user_id and expiration (e.g., 1 hour)
  - Send email with reset link (or just log for now if email not set up)
  - Return: { success: true, message: "If an account exists, a reset link has been sent" }
- If user doesn't exist:
  - Return same message (don't reveal if email exists for security)
  - Return: { success: true, message: "If an account exists, a reset link has been sent" }

For now, you can just log the reset token to console if email sending isn't configured.

Priority 2: Security Enhancement

Prompt 13: Move Tokens to httpOnly Cookies (Advanced)

This is a larger change that affects both frontend and backend:

BACKEND changes:
1. In the login endpoint, instead of returning the token in JSON, set it as an httpOnly cookie:

```
response = jsonify({ success: true, user_id: ..., email: ..., name: ... })
response.set_cookie(
  'mingus_token',
  token,
  httponly=True,
  secure=True,  # Set to False for local development
  samesite='Lax',
  max_age=expiry_seconds
)
```

return response

2. Update the auth middleware to read from cookies instead of Authorization header:
token = request.cookies.get('mingus_token')

3. Add a logout endpoint that clears the cookie:
POST /api/auth/logout
response.delete_cookie('mingus_token')

FRONTEND changes:
1. Remove localStorage.setItem('mingus_token', ...) from login
2. Remove localStorage.getItem('mingus_token') from API calls
3. Add credentials: 'include' to all fetch calls
4. Remove the Authorization header from fetch calls
5. Update the logout function to call POST /api/auth/logout instead of just clearing localStorage

Note: This requires CORS configuration to allow credentials. Update Flask-CORS:
CORS(app, supports_credentials=True, origins=['http://localhost:5173'])

Testing Prompts

Prompt 14: Verify Landing Page Changes

Review the landing page implementation and verify:

1. The HeroSection assessment buttons scroll to the #assessments section
2. There is only ONE assessment section on the page (the unified one with id="assessments")
3. The AssessmentSection component import and usage have been removed
4. All 4 assessments are displayed in the unified section
5. Each assessment card shows: icon, FREE badge, title, description, time estimate
6. The "Skip assessments" link works
7. Clicking an assessment card opens the AssessmentModal correctly

List any issues found.

Prompt 15: Verify User Flow Changes

Review the user registration and login flow and verify:

1. The signup form has a single password field with show/hide toggle
2. There is no password confirmation field
3. The QuickSetupOverlay component exists and is integrated into the dashboard
4. After registration, users go directly to the dashboard (not a separate setup page)
5. The login form has a "Remember Me" checkbox
6. The login form has a "Forgot password?" link
7. The forgot password page exists and is routed correctly

List any issues found or incomplete implementations.

Order of Implementation

For best results, run these prompts in this order:

Phase 1 - Landing Page (Do First)

1. Prompt 1 (Anchor navigation)
2. Prompt 2 (Unified assessment section)
3. Prompt 3 (Remove duplicate)

Phase 2 - Registration Flow 4. Prompt 4 (Password toggle) 5. Prompt 5 (QuickSetupOverlay) 6. Prompt 6 (Integrate overlay) 7. Prompt 7 (Update navigation)

Phase 3 - Login Flow 8. Prompt 8 (Remember Me) 9. Prompt 9 (Forgot Password page)

Phase 4 - Backend 10. Prompt 10 (Login Remember Me) 11. Prompt 11 (Quick Setup endpoints) 12. Prompt 12 (Forgot Password endpoint)

Phase 5 - Security (Optional) 13. Prompt 13 (httpOnly cookies)

Phase 6 - Testing 14. Prompt 14 (Landing page verification) 15. Prompt 15 (User flow verification)

Tips for Using These Prompts with Cursor

1. **Open relevant files first** - Before pasting a prompt, open the files mentioned in Cursor so it has context
2. **Use @file references** - You can reference files in Cursor prompts like: `@frontend/src/components/LandingPage.tsx`
3. **Review changes before accepting** - Always review Cursor's suggested changes before applying them
4. **Run the app after each phase** - Test the changes work before moving to the next phase
5. **Commit after each phase** - Use git commits so you can rollback if something breaks

6. **Adjust paths as needed** - Your file structure may differ slightly from what's assumed here

---

Document prepared: January 2025 For: Mingus Personal Finance Application