MINGUS Data Collection Implementation Guide

🚀 Step-by-Step Implementation in Cursor

1. Database Schema Updates

First, update your database schema to include the new fields:

```sql
```

```sql
-- Add new columns to users table
ALTER TABLE users ADD COLUMN first_name VARCHAR(50);
ALTER TABLE users ADD COLUMN last_name VARCHAR(50);
ALTER TABLE users ADD COLUMN date_of_birth DATE;
ALTER TABLE users ADD COLUMN zip_code VARCHAR(10);
ALTER TABLE users ADD COLUMN phone_number VARCHAR(20);
ALTER TABLE users ADD COLUMN email_verification_status BOOLEAN DEFAULT FALSE;

-- Financial data
ALTER TABLE users ADD COLUMN monthly_income DECIMAL(10,2);
ALTER TABLE users ADD COLUMN income_frequency ENUM('weekly', 'bi-weekly', 'semi-monthly', 'monthly', 'ann
ALTER TABLE users ADD COLUMN primary_income_source VARCHAR(100);
ALTER TABLE users ADD COLUMN current_savings_balance DECIMAL(10,2);
ALTER TABLE users ADD COLUMN total_debt_amount DECIMAL(10,2);
ALTER TABLE users ADD COLUMN credit_score_range ENUM('excellent', 'good', 'fair', 'poor', 'very_poor', 'unknov
ALTER TABLE users ADD COLUMN employment_status VARCHAR(50);

-- Demographics
ALTER TABLE users ADD COLUMN age_range ENUM('18-24', '25-34', '35-44', '45-54', '55-64', '65+');
ALTER TABLE users ADD COLUMN marital_status ENUM('single', 'married', 'partnership', 'divorced', 'widowed', 'p
ALTER TABLE users ADD COLUMN dependents_count INT DEFAULT 0;
ALTER TABLE users ADD COLUMN household_size INT DEFAULT 1;
ALTER TABLE users ADD COLUMN education_level VARCHAR(100);
ALTER TABLE users ADD COLUMN occupation VARCHAR(100);
ALTER TABLE users ADD COLUMN industry VARCHAR(100);
ALTER TABLE users ADD COLUMN years_of_experience ENUM('less_than_1', '1-3', '4-7', '8-12', '13-20', '20+');

-- Goals and preferences
ALTER TABLE users ADD COLUMN primary_financial_goal VARCHAR(100);
ALTER TABLE users ADD COLUMN risk_tolerance_level ENUM('conservative', 'moderate', 'aggressive', 'unsure');
ALTER TABLE users ADD COLUMN financial_knowledge_level ENUM('beginner', 'intermediate', 'advanced', 'exper
ALTER TABLE users ADD COLUMN preferred_contact_method VARCHAR(50);
ALTER TABLE users ADD COLUMN notification_preferences JSON;

-- Health and wellness
ALTER TABLE users ADD COLUMN health_checkin_frequency ENUM('daily', 'weekly', 'monthly', 'on_demand', 'ne
ALTER TABLE users ADD COLUMN stress_level_baseline INT CHECK (stress_level_baseline >= 1 AND stress_leve
ALTER TABLE users ADD COLUMN wellness_goals JSON;

-- Compliance and preferences
ALTER TABLE users ADD COLUMN gdpr_consent_status BOOLEAN DEFAULT FALSE;
ALTER TABLE users ADD COLUMN data_sharing_preferences VARCHAR(100);
```

```
ALTER TABLE users ADD COLUMN profile_completion_percentage DECIMAL(5,2) DEFAULT 0.00;
ALTER TABLE users ADD COLUMN onboarding_step INT DEFAULT 1;
```

2. Create TypeScript Interfaces

Create type definitions for your new data structures:

```typescript
```

```typescript
// types/user.ts
export interface UserProfile {
  // Authentication & Profile
  id: string;
  email: string;
  firstName?: string;
  lastName?: string;
  dateOfBirth?: Date;
  zipCode?: string;
  phoneNumber?: string;
  emailVerificationStatus: boolean;

  // Financial Data
  monthlyIncome?: number;
  incomeFrequency?: 'weekly' | 'bi-weekly' | 'semi-monthly' | 'monthly' | 'annually';
  primaryIncomeSource?: string;
  currentSavingsBalance?: number;
  totalDebtAmount?: number;
  creditScoreRange?: 'excellent' | 'good' | 'fair' | 'poor' | 'very_poor' | 'unknown';
  employmentStatus?: string;

  // Demographics
  ageRange?: '18-24' | '25-34' | '35-44' | '45-54' | '55-64' | '65+';
  maritalStatus?: 'single' | 'married' | 'partnership' | 'divorced' | 'widowed' | 'prefer_not_to_say';
  dependentsCount?: number;
  householdSize?: number;
  educationLevel?: string;
  occupation?: string;
  industry?: string;
  yearsOfExperience?: 'less_than_1' | '1-3' | '4-7' | '8-12' | '13-20' | '20+';

  // Goals & Preferences
  primaryFinancialGoal?: string;
  riskToleranceLevel?: 'conservative' | 'moderate' | 'aggressive' | 'unsure';
  financialKnowledgeLevel?: 'beginner' | 'intermediate' | 'advanced' | 'expert';
  preferredContactMethod?: string;
  notificationPreferences?: NotificationPreferences;

  // Health & Wellness
  healthCheckinFrequency?: 'daily' | 'weekly' | 'monthly' | 'on_demand' | 'never';
  stressLevelBaseline?: number;
  wellnessGoals?: string[];
```

```
  // Meta
  profileCompletionPercentage: number;
  onboardingStep: number;
  gdprConsentStatus: boolean;
  dataSharingPreferences?: string;
}

export interface NotificationPreferences {
  weeklyInsights: boolean;
  monthlySpendingSummaries: boolean;
  goalProgressUpdates: boolean;
  billPaymentReminders: boolean;
  marketUpdates: boolean;
  educationalContent: boolean;
  productUpdates: boolean;
}

export interface OnboardingStep {
  step: number;
  title: string;
  subtitle?: string;
  fields: FormField[];
  isRequired: boolean;
  category: 'critical' | 'important' | 'enhanced';
}

export interface FormField {
  name: string;
  type: 'text' | 'email' | 'tel' | 'date' | 'number' | 'select' | 'checkbox' | 'radio' | 'slider' | 'currency';
  label: string;
  placeholder?: string;
  subtitle?: string;
  required: boolean;
  options?: { value: string; label: string }[];
  validation?: {
    min?: number;
    max?: number;
    pattern?: string;
    message?: string;
  };
}
```

3. Create Onboarding Configuration

Define your onboarding flow configuration:

```typescript
```

```
// config/onboarding.ts
import { OnboardingStep } from '../types/user';

export const ONBOARDING_STEPS: OnboardingStep[] = [
 {
  step: 1,
  title: "Let's get to know you",
  subtitle: "We'll start with some basic information",
  category: 'critical',
  isRequired: true,
  fields: [
   {
    name: 'firstName',
    type: 'text',
    label: "What's your first name?",
    placeholder: 'Enter your first name',
    required: true,
    validation: {
     min: 2,
     max: 50,
     pattern: '^[a-zA-Z]+$',
     message: 'Please enter a valid first name'
    }
   },
   {
    name: 'lastName',
    type: 'text',
    label: "What's your last name?",
    placeholder: 'Enter your last name',
    required: true,
    validation: {
     min: 2,
     max: 50,
     pattern: '^[a-zA-Z]+$',
     message: 'Please enter a valid last name'
    }
   },
   {
    name: 'dateOfBirth',
    type: 'date',
    label: 'When were you born?',
    subtitle: 'We use this to provide age-appropriate financial advice',
    required: true,
```

```
      validation: {
        message: 'You must be 18 or older to use this service'
      }
    },
    {
      name: 'zipCode',
      type: 'text',
      label: "What's your ZIP code?",
      subtitle: 'This helps us provide location-specific financial insights',
      placeholder: '12345',
      required: true,
      validation: {
        pattern: '^\\d{5}$',
        message: 'Please enter a valid 5-digit ZIP code'
      }
    },
    {
      name: 'phoneNumber',
      type: 'tel',
      label: "What's your phone number?",
      subtitle: 'For account security and important notifications',
      placeholder: '(555) 123-4567',
      required: true
    }
  ]
},
{
  step: 2,
  title: "Tell us about your finances",
  subtitle: "This helps us provide personalized insights",
  category: 'critical',
  isRequired: true,
  fields: [
    {
      name: 'monthlyIncome',
      type: 'currency',
      label: "What's your monthly income before taxes?",
      subtitle: 'Include all sources of income',
      placeholder: '$5,000',
      required: true,
      validation: {
        min: 0,
        max: 999999,
        message: 'Please enter a valid income amount'
```

```
  }
},
{
  name: 'incomeFrequency',
  type: 'select',
  label: 'How often do you receive income?',
  required: true,
  options: [
    { value: 'weekly', label: 'Weekly' },
    { value: 'bi-weekly', label: 'Bi-weekly (every 2 weeks)' },
    { value: 'semi-monthly', label: 'Semi-monthly (twice a month)' },
    { value: 'monthly', label: 'Monthly' },
    { value: 'annually', label: 'Annually' }
  ]
},
{
  name: 'primaryIncomeSource',
  type: 'select',
  label: "What's your primary source of income?",
  required: true,
  options: [
    { value: 'full-time-employment', label: 'Full-time employment' },
    { value: 'part-time-employment', label: 'Part-time employment' },
    { value: 'self-employment', label: 'Self-employment/Freelancing' },
    { value: 'business-ownership', label: 'Business ownership' },
    { value: 'investment-income', label: 'Investment income' },
    { value: 'retirement', label: 'Retirement/Pension' },
    { value: 'government-benefits', label: 'Government benefits' },
    { value: 'other', label: 'Other' }
  ]
},
{
  name: 'employmentStatus',
  type: 'select',
  label: "What's your current employment status?",
  required: true,
  options: [
    { value: 'employed-full-time', label: 'Employed full-time' },
    { value: 'employed-part-time', label: 'Employed part-time' },
    { value: 'self-employed', label: 'Self-employed' },
    { value: 'unemployed', label: 'Unemployed' },
    { value: 'student', label: 'Student' },
    { value: 'retired', label: 'Retired' },
    { value: 'unable-to-work', label: 'Unable to work' }
```

```
      ]
    }
  ]
},
{
  step: 3,
  title: "Your financial picture",
  subtitle: "Help us understand your current financial situation",
  category: 'critical',
  isRequired: true,
  fields: [
    {
      name: 'currentSavingsBalance',
      type: 'currency',
      label: 'How much do you currently have in savings?',
      subtitle: 'Include checking, savings, and money market accounts',
      placeholder: '$10,000',
      required: false
    },
    {
      name: 'totalDebtAmount',
      type: 'currency',
      label: "What's your total debt amount?",
      subtitle: 'Include credit cards, loans, mortgage, etc.',
      placeholder: '$25,000',
      required: false
    },
    {
      name: 'creditScoreRange',
      type: 'select',
      label: "What's your approximate credit score?",
      required: false,
      options: [
        { value: 'excellent', label: 'Excellent (750+)' },
        { value: 'good', label: 'Good (700-749)' },
        { value: 'fair', label: 'Fair (650-699)' },
        { value: 'poor', label: 'Poor (600-649)' },
        { value: 'very_poor', label: 'Very Poor (Below 600)' },
        { value: 'unknown', label: "I don't know" }
      ]
    }
  ]
},
```

```
  // Add more steps for demographics, goals, etc.
];
```

4. Create React Components

Build reusable form components:

```tsx
```

```tsx
// components/forms/FormField.tsx
import React from 'react';
import { FormField as FormFieldType } from '../../types/user';

interface FormFieldProps {
  field: FormFieldType;
  value: any;
  onChange: (name: string, value: any) => void;
  error?: string;
}

export const FormField: React.FC<FormFieldProps> = ({ field, value, onChange, error }) => {
  const handleChange = (e: React.ChangeEvent<HTMLInputElement | HTMLSelectElement>) => {
    let newValue = e.target.value;

    if (field.type === 'number' || field.type === 'currency') {
      newValue = parseFloat(newValue) || 0;
    }

    onChange(field.name, newValue);
  };

  const renderInput = () => {
    switch (field.type) {
      case 'select':
        return (
          <select
            id={field.name}
            value={value || ''}
            onChange={handleChange}
            className="w-full p-3 border rounded-lg focus:ring-2 focus:ring-blue-500"
            required={field.required}
          >
            <option value="">Select an option...</option>
            {field.options?.map((option) => (
              <option key={option.value} value={option.value}>
                {option.label}
              </option>
            ))}
          </select>
        );

      case 'currency':
```

```
      return (
        <div className="relative">
          <span className="absolute left-3 top-3 text-gray-500">$</span>
          <input
            type="number"
            id={field.name}
            value={value || ''}
            onChange={handleChange}
            placeholder={field.placeholder?.replace('$', '')}
            className="w-full p-3 pl-8 border rounded-lg focus:ring-2 focus:ring-blue-500"
            required={field.required}
            min={field.validation?.min}
            max={field.validation?.max}
          />
        </div>
      );

    case 'slider':
      return (
        <div className="space-y-2">
          <input
            type="range"
            id={field.name}
            value={value || field.validation?.min || 1}
            onChange={handleChange}
            min={field.validation?.min || 1}
            max={field.validation?.max || 10}
            className="w-full h-2 bg-gray-200 rounded-lg appearance-none cursor-pointer"
          />
          <div className="flex justify-between text-sm text-gray-500">
            <span>{field.validation?.min || 1}</span>
            <span className="font-medium">{value || field.validation?.min || 1}</span>
            <span>{field.validation?.max || 10}</span>
          </div>
        </div>
      );

    default:
      return (
        <input
          type={field.type}
          id={field.name}
          value={value || ''}
          onChange={handleChange}
```

```
          placeholder={field.placeholder}
          className="w-full p-3 border rounded-lg focus:ring-2 focus:ring-blue-500"
          required={field.required}
          pattern={field.validation?.pattern}
        />
      );
    }
  };

  return (
    <div className="space-y-2">
      <label htmlFor={field.name} className="block text-lg font-medium text-gray-900">
        {field.label}
        {field.required && <span className="text-red-500 ml-1">*</span>}
      </label>

      {field.subtitle && (
        <p className="text-sm text-gray-600">{field.subtitle}</p>
      )}

      {renderInput()}

      {error && (
        <p className="text-sm text-red-600">{error}</p>
      )}
    </div>
  );
};
```

tsx

```tsx
// components/onboarding/OnboardingStep.tsx
import React, { useState } from 'react';
import { OnboardingStep as OnboardingStepType } from '../../types/user';
import { FormField } from '../forms/FormField';

interface OnboardingStepProps {
  step: OnboardingStepType;
  data: Record<string, any>;
  onNext: (data: Record<string, any>) => void;
  onPrevious: () => void;
  isFirst: boolean;
  isLast: boolean;
}

export const OnboardingStep: React.FC<OnboardingStepProps> = ({
  step,
  data,
  onNext,
  onPrevious,
  isFirst,
  isLast
}) => {
  const [formData, setFormData] = useState(data);
  const [errors, setErrors] = useState<Record<string, string>>({});

  const handleFieldChange = (name: string, value: any) => {
    setFormData(prev => ({ ...prev, [name]: value }));

    // Clear error when user starts typing
    if (errors[name]) {
      setErrors(prev => ({ ...prev, [name]: '' }));
    }
  };

  const validateForm = () => {
    const newErrors: Record<string, string> = {};

    step.fields.forEach(field => {
      const value = formData[field.name];

      if (field.required && (!value || value === '')) {
        newErrors[field.name] = `${field.label} is required`;
        return;
```

```
    }

    if (field.validation) {
      const { min, max, pattern, message } = field.validation;

      if (min && value < min) {
        newErrors[field.name] = message || `Minimum value is ${min}`;
      }

      if (max && value > max) {
        newErrors[field.name] = message || `Maximum value is ${max}`;
      }

      if (pattern && typeof value === 'string' && !new RegExp(pattern).test(value)) {
        newErrors[field.name] = message || 'Invalid format';
      }
    }
  });

  setErrors(newErrors);
  return Object.keys(newErrors).length === 0;
};

const handleNext = () => {
  if (validateForm()) {
    onNext(formData);
  }
};

return (
  <div className="max-w-2xl mx-auto p-6">
    <div className="mb-8">
      <h2 className="text-3xl font-bold text-gray-900 mb-2">{step.title}</h2>
      {step.subtitle && (
        <p className="text-lg text-gray-600">{step.subtitle}</p>
      )}
    </div>

    <form className="space-y-6" onSubmit={(e) => { e.preventDefault(); handleNext(); }}>
      {step.fields.map((field) => (
        <FormField
          key={field.name}
          field={field}
          value={formData[field.name]}
```

```
          onChange={handleFieldChange}
          error={errors[field.name]}
        />
      ))}

      <div className="flex justify-between pt-6">
        {!isFirst && (
          <button
            type="button"
            onClick={onPrevious}
            className="px-6 py-3 border border-gray-300 rounded-lg hover:bg-gray-50"
          >
            Previous
          </button>
        )}

        <button
          type="submit"
          className="px-6 py-3 bg-blue-600 text-white rounded-lg hover:bg-blue-700 ml-auto"
        >
          {isLast ? 'Complete Setup' : 'Next'}
        </button>
      </div>
    </form>
  </div>
 );
};
```

5. Create the Main Onboarding Flow

tsx

```tsx
// components/onboarding/OnboardingFlow.tsx
import React, { useState, useEffect } from 'react';
import { OnboardingStep } from './OnboardingStep';
import { ONBOARDING_STEPS } from '../../config/onboarding';
import { UserProfile } from '../../types/user';

interface OnboardingFlowProps {
  onComplete: (userData: Partial<UserProfile>) => void;
  initialData?: Partial<UserProfile>;
}

export const OnboardingFlow: React.FC<OnboardingFlowProps> = ({
  onComplete,
  initialData = {}
}) => {
  const [currentStep, setCurrentStep] = useState(0);
  const [userData, setUserData] = useState<Partial<UserProfile>>(initialData);

  const handleNext = (stepData: Record<string, any>) => {
    const updatedData = { ...userData, ...stepData };
    setUserData(updatedData);

    if (currentStep < ONBOARDING_STEPS.length - 1) {
      setCurrentStep(currentStep + 1);
    } else {
      // Calculate completion percentage
      const totalFields = ONBOARDING_STEPS.flatMap(step => step.fields).length;
      const completedFields = Object.keys(updatedData).length;
      const completionPercentage = (completedFields / totalFields) * 100;

      onComplete({
        ...updatedData,
        profileCompletionPercentage: completionPercentage,
        onboardingStep: ONBOARDING_STEPS.length
      });
    }
  };

  const handlePrevious = () => {
    if (currentStep > 0) {
      setCurrentStep(currentStep - 1);
    }
  };
```

```
  const currentStepData = ONBOARDING_STEPS[currentStep];

  return (
    <div className="min-h-screen bg-gray-50 py-12">
      {/* Progress Bar */}
      <div className="max-w-2xl mx-auto mb-8 px-6">
        <div className="flex justify-between items-center mb-4">
          <span className="text-sm font-medium text-gray-600">
            Step {currentStep + 1} of {ONBOARDING_STEPS.length}
          </span>
          <span className="text-sm text-gray-600">
            {Math.round(((currentStep + 1) / ONBOARDING_STEPS.length) * 100)}% complete
          </span>
        </div>
        <div className="w-full bg-gray-200 rounded-full h-2">
          <div
            className="bg-blue-600 h-2 rounded-full transition-all duration-300"
            style={{ width: `${((currentStep + 1) / ONBOARDING_STEPS.length) * 100}%` }}
          />
        </div>
      </div>

      <OnboardingStep
        step={currentStepData}
        data={userData}
        onNext={handleNext}
        onPrevious={handlePrevious}
        isFirst={currentStep === 0}
        isLast={currentStep === ONBOARDING_STEPS.length - 1}
      />
    </div>
  );
};
```

6. API Integration

Create API endpoints for saving user data:

typescript

```typescript
// pages/api/user/profile.ts (Next.js example)
import { NextApiRequest, NextApiResponse } from 'next';
import { getUserFromSession, updateUserProfile } from '../../../lib/auth';
import { validateUserProfileData } from '../../../lib/validation';

export default async function handler(req: NextApiRequest, res: NextApiResponse) {
  if (req.method !== 'PATCH') {
    return res.status(405).json({ message: 'Method not allowed' });
  }

  try {
    const user = await getUserFromSession(req);
    if (!user) {
      return res.status(401).json({ message: 'Unauthorized' });
    }

    const validation = validateUserProfileData(req.body);
    if (!validation.isValid) {
      return res.status(400).json({
        message: 'Validation failed',
        errors: validation.errors
      });
    }

    const updatedUser = await updateUserProfile(user.id, req.body);

    res.status(200).json({
      message: 'Profile updated successfully',
      user: updatedUser
    });
  } catch (error) {
    console.error('Profile update error:', error);
    res.status(500).json({ message: 'Internal server error' });
  }
}
```

typescript

```typescript
// lib/validation.ts
import { UserProfile } from '../types/user';

export function validateUserProfileData(data: Partial<UserProfile>) {
  const errors: string[] = [];

  // Validate required fields
  if (data.firstName && data.firstName.length < 2) {
    errors.push('First name must be at least 2 characters');
  }

  if (data.email && !/^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(data.email)) {
    errors.push('Invalid email format');
  }

  if (data.zipCode && !/^\d{5}$/.test(data.zipCode)) {
    errors.push('ZIP code must be 5 digits');
  }

  if (data.monthlyIncome && (data.monthlyIncome < 0 || data.monthlyIncome > 999999)) {
    errors.push('Monthly income must be between $0 and $999,999');
  }

  return {
    isValid: errors.length === 0,
    errors
  };
}
```

7. State Management (Optional - using Zustand)

typescript

```ts
// store/userStore.ts
import { create } from 'zustand';
import { UserProfile } from '../types/user';

interface UserStore {
  user: Partial<UserProfile>;
  isLoading: boolean;
  updateUser: (data: Partial<UserProfile>) => void;
  saveProfile: () => Promise<void>;
}

export const useUserStore = create<UserStore>((set, get) => ({
  user: {},
  isLoading: false,

  updateUser: (data) => set((state) => ({
    user: { ...state.user, ...data }
  })),

  saveProfile: async () => {
    set({ isLoading: true });
    try {
      const { user } = get();
      const response = await fetch('/api/user/profile', {
        method: 'PATCH',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(user),
      });

      if (!response.ok) throw new Error('Failed to save profile');

      const result = await response.json();
      set({ user: result.user });
    } catch (error) {
      console.error('Failed to save profile:', error);
    } finally {
      set({ isLoading: false });
    }
  },
}));
```

🎯 Usage in Your App

1. **Replace your current signup form** with the OnboardingFlow component
2. **Add to your main app component:**

```tsx
// pages/onboarding.tsx
import { OnboardingFlow } from '../components/onboarding/OnboardingFlow';
import { useRouter } from 'next/router';

export default function OnboardingPage() {
  const router = useRouter();

  const handleComplete = async (userData) => {
    try {
      // Save to your backend
      await fetch('/api/user/profile', {
        method: 'PATCH',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(userData),
      });

      // Redirect to dashboard
      router.push('/dashboard');
    } catch (error) {
      console.error('Failed to save user data:', error);
    }
  };

  return <OnboardingFlow onComplete={handleComplete} />;
}
```

🔧 Cursor-Specific Tips

1. **Use Cursor's autocomplete** - The TypeScript interfaces will provide excellent IntelliSense
2. **Leverage Cursor's refactoring** - Select code and ask Cursor to refactor or optimize
3. **Use Cursor's AI chat** to:
   - Generate additional validation rules
   - Create test cases for your forms
   - Optimize component performance
   - Generate additional form field types

📝 Next Steps

1. Run database migrations to add new columns
2. Create the TypeScript interfaces in your project
3. Build the form components step by step
4. Test the onboarding flow
5. Add analytics to track completion rates
6. Implement progressive enhancement for optional fields

This implementation provides a solid foundation that you can customize and expand based on your specific needs.