Perfect! I've created a comprehensive implementation of the recommended improvements for your Mingus onboarding flow. Here's what this provides:

**🎯 Key Improvements Implemented:**

**1. Centralized OnboardingFlowService**

* **Single source of truth** for step definitions, navigation logic, and backend communication
* **Backend-driven progress** that fetches real-time status from your progress tracking API
* **Smart step validation** that checks dependencies and prerequisites
* **Universal navigation** that determines next steps based on backend state

**2. OnboardingFlowGuard Component**

* **Route protection** that prevents users from accessing steps they shouldn't
* **Automatic redirects** to the correct step if users try to skip ahead
* **Loading states** while checking permissions
* **Error handling** with fallback to welcome step

**3. UniversalContinueButton**

* **Backend integration** that marks steps complete via your API
* **Automatic navigation** to the next correct step (no hardcoded routes)
* **Loading states** and error handling
* **Validation** that ensures steps can't be completed with incomplete data

**4. Enhanced Progress Tracking**

* **Visual progress bar** showing completion percentage
* **Step indicators** with current/completed/accessible states
* **Time estimates** based on your step definitions
* **Real-time updates** as users complete steps

**🔄 How It Works With Your System:**

1. **User tries to access /onboarding/financial**
2. **OnboardingFlowGuard** calls your backend API to check progress
3. **If prerequisites not met** (e.g., profile\_setup not complete), redirects to correct step
4. **If accessible**, renders the step component
5. **User fills out form** and clicks UniversalContinueButton
6. **Button calls your API** to mark step complete with completion data
7. **Backend returns next step**, button navigates automatically
8. **Progress bar updates** to reflect new completion status

**🛠 Integration with Your Backend:**

The service integrates perfectly with your Python progress tracking template:

* Calls /api/onboarding/progress/{userId} to get current status
* Calls /api/onboarding/complete-step to mark steps complete
* Uses your completion criteria validation
* Respects your step dependencies and weights

**🚀 Next Steps:**

1. **Update your existing step components** to use OnboardingFlowGuard and UniversalContinueButton
2. **Replace hardcoded navigation** with the flow service
3. **Test the flow** by trying to skip steps - you should be redirected automatically
4. **Add error boundaries** for production robustness

This implementation makes your onboarding flow robust, backend-driven, and user-proof. Users can only proceed in the correct order, and the system gracefully handles edge cases like page refreshes, direct URL access, and incomplete data.

Would you like me to show you how to implement this in a specific step component, or help you adapt any existing components?

Retry

[Claude can make mistakes.   
Please double-check responses.](https://support.anthropic.com/en/articles/8525154-claude-is-providing-incorrect-or-misleading-responses-what-s-going-on)

Research

Sonnet 4

// src/types/onboarding.ts

export interface OnboardingStep {

id: string;

name: string;

title: string;

description: string;

route: string;

stepNumber: number;

required: boolean;

estimatedTime: number;

weight: number;

completionCriteria: string[];

dependencies?: string[];

}

export interface OnboardingProgress {

userId: string;

currentStep: string;

completedSteps: string[];

progressPercentage: number;

isCompleted: boolean;

stepCompletionData: Record<string, any>;

estimatedTimeRemaining: number;

createdAt: string;

updatedAt: string;

}

export interface StepCompletionResult {

success: boolean;

stepCompleted: string;

newProgressPercentage: number;

nextStep: string | null;

isOnboardingComplete: boolean;

error?: string;

}

export interface NavigationContext {

currentStep: string;

nextStep: string | null;

canProceed: boolean;

canAccess: boolean;

redirectTo?: string;

}

// src/services/OnboardingFlowService.ts

import { OnboardingStep, OnboardingProgress, StepCompletionResult, NavigationContext } from '../types/onboarding';

export class OnboardingFlowService {

private static instance: OnboardingFlowService;

private steps: OnboardingStep[];

private baseUrl = '/api/onboarding';

private constructor() {

this.steps = this.defineSteps();

}

public static getInstance(): OnboardingFlowService {

if (!OnboardingFlowService.instance) {

OnboardingFlowService.instance = new OnboardingFlowService();

}

return OnboardingFlowService.instance;

}

private defineSteps(): OnboardingStep[] {

return [

{

id: 'welcome',

name: 'welcome',

title: 'Welcome & Introduction',

description: 'Introduction to Mingus and onboarding process explanation',

route: '/onboarding/welcome',

stepNumber: 1,

required: true,

estimatedTime: 2,

weight: 5,

completionCriteria: ['viewed\_welcome\_screen', 'clicked\_get\_started']

},

{

id: 'profile\_setup',

name: 'profile\_setup',

title: 'Basic Profile Setup',

description: 'Name, age, location, employment status',

route: '/onboarding/profile',

stepNumber: 2,

required: true,

estimatedTime: 3,

weight: 10,

completionCriteria: ['name\_provided', 'age\_provided', 'location\_provided', 'employment\_status\_provided'],

dependencies: ['welcome']

},

{

id: 'financial\_profile',

name: 'financial\_profile',

title: 'Financial Profile',

description: 'Income sources, monthly expenses, current debt, savings',

route: '/onboarding/financial',

stepNumber: 3,

required: true,

estimatedTime: 5,

weight: 20,

completionCriteria: ['income\_provided', 'expenses\_provided', 'debt\_info\_provided', 'savings\_info\_provided'],

dependencies: ['profile\_setup']

},

{

id: 'goals\_setup',

name: 'goals\_setup',

title: 'Financial Goals Setting',

description: 'Short-term and long-term financial objectives',

route: '/onboarding/goals',

stepNumber: 4,

required: true,

estimatedTime: 4,

weight: 15,

completionCriteria: ['short\_term\_goals\_set', 'long\_term\_goals\_set', 'timeline\_provided'],

dependencies: ['financial\_profile']

},

{

id: 'health\_assessment',

name: 'health\_assessment',

title: 'Health & Wellness Assessment',

description: 'Initial stress levels, physical activity, mental health baseline',

route: '/onboarding/health',

stepNumber: 5,

required: true,

estimatedTime: 3,

weight: 15,

completionCriteria: ['stress\_level\_provided', 'activity\_level\_provided', 'wellness\_baseline\_set'],

dependencies: ['goals\_setup']

},

{

id: 'data\_input',

name: 'data\_input',

title: 'Financial Data Integration',

description: 'Connect accounts or manual data entry',

route: '/onboarding/data-input',

stepNumber: 6,

required: true,

estimatedTime: 7,

weight: 20,

completionCriteria: ['data\_source\_selected', 'accounts\_connected\_or\_manual\_complete', 'expense\_categories\_set'],

dependencies: ['health\_assessment']

},

{

id: 'preferences',

name: 'preferences',

title: 'Preferences & Settings',

description: 'Risk tolerance, notification preferences, privacy settings',

route: '/onboarding/preferences',

stepNumber: 7,

required: false,

estimatedTime: 3,

weight: 10,

completionCriteria: ['risk\_tolerance\_set', 'notifications\_configured', 'privacy\_settings\_set'],

dependencies: ['data\_input']

},

{

id: 'dashboard\_intro',

name: 'dashboard\_intro',

title: 'Dashboard Introduction',

description: 'Tour of main features and dashboard walkthrough',

route: '/onboarding/dashboard-intro',

stepNumber: 8,

required: true,

estimatedTime: 3,

weight: 5,

completionCriteria: ['dashboard\_tour\_completed', 'first\_forecast\_viewed', 'help\_resources\_acknowledged'],

dependencies: ['data\_input'] // Can skip preferences

}

];

}

// Backend API calls

async fetchProgress(userId: string): Promise<OnboardingProgress> {

try {

const response = await fetch(`${this.baseUrl}/progress/${userId}`, {

method: 'GET',

headers: {

'Content-Type': 'application/json',

'Authorization': `Bearer ${localStorage.getItem('token')}`

}

});

if (!response.ok) {

throw new Error(`Failed to fetch progress: ${response.statusText}`);

}

return await response.json();

} catch (error) {

console.error('Error fetching onboarding progress:', error);

throw error;

}

}

async completeStep(userId: string, stepName: string, completionData: Record<string, any>): Promise<StepCompletionResult> {

try {

const response = await fetch(`${this.baseUrl}/complete-step`, {

method: 'POST',

headers: {

'Content-Type': 'application/json',

'Authorization': `Bearer ${localStorage.getItem('token')}`

},

body: JSON.stringify({

user\_id: userId,

step\_name: stepName,

completion\_data: completionData

})

});

if (!response.ok) {

throw new Error(`Failed to complete step: ${response.statusText}`);

}

return await response.json();

} catch (error) {

console.error('Error completing step:', error);

throw error;

}

}

// Navigation Logic

async getNavigationContext(userId: string, requestedStep: string): Promise<NavigationContext> {

const progress = await this.fetchProgress(userId);

const step = this.getStepById(requestedStep);

if (!step) {

return {

currentStep: requestedStep,

nextStep: null,

canProceed: false,

canAccess: false,

redirectTo: '/onboarding/welcome'

};

}

const canAccess = this.canAccessStep(requestedStep, progress);

const nextStep = this.getNextStep(progress);

// If trying to access a step they can't access, redirect to correct step

if (!canAccess) {

const correctStep = this.getCurrentAccessibleStep(progress);

return {

currentStep: requestedStep,

nextStep,

canProceed: false,

canAccess: false,

redirectTo: correctStep.route

};

}

return {

currentStep: requestedStep,

nextStep,

canProceed: true,

canAccess: true

};

}

async proceedToNextStep(userId: string, currentStep: string, completionData: Record<string, any>): Promise<{ route: string; isComplete: boolean }> {

// Complete current step

const result = await this.completeStep(userId, currentStep, completionData);

if (!result.success) {

throw new Error(result.error || 'Failed to complete step');

}

// If onboarding is complete

if (result.isOnboardingComplete) {

return { route: '/dashboard', isComplete: true };

}

// Get next step route

if (result.nextStep) {

const nextStepInfo = this.getStepById(result.nextStep);

return { route: nextStepInfo?.route || '/dashboard', isComplete: false };

}

return { route: '/dashboard', isComplete: true };

}

// Utility methods

getStepById(stepId: string): OnboardingStep | undefined {

return this.steps.find(step => step.id === stepId);

}

getStepByRoute(route: string): OnboardingStep | undefined {

return this.steps.find(step => step.route === route);

}

canAccessStep(stepId: string, progress: OnboardingProgress): boolean {

const step = this.getStepById(stepId);

if (!step) return false;

// Check if all dependencies are completed

if (step.dependencies) {

return step.dependencies.every(dep => progress.completedSteps.includes(dep));

}

return true;

}

getNextStep(progress: OnboardingProgress): string | null {

// Find first incomplete step that user can access

for (const step of this.steps) {

if (!progress.completedSteps.includes(step.id) && this.canAccessStep(step.id, progress)) {

return step.id;

}

}

return null;

}

getCurrentAccessibleStep(progress: OnboardingProgress): OnboardingStep {

const nextStep = this.getNextStep(progress);

if (nextStep) {

return this.getStepById(nextStep)!;

}

// If no next step, return the last step (dashboard intro) or welcome

return this.steps[this.steps.length - 1] || this.steps[0];

}

getAllSteps(): OnboardingStep[] {

return [...this.steps];

}

getProgressSummary(progress: OnboardingProgress) {

const totalSteps = this.steps.filter(s => s.required).length;

const completedRequiredSteps = progress.completedSteps.filter(stepId => {

const step = this.getStepById(stepId);

return step?.required;

}).length;

return {

totalSteps,

completedSteps: completedRequiredSteps,

progressPercentage: progress.progressPercentage,

estimatedTimeRemaining: progress.estimatedTimeRemaining,

currentStep: this.getStepById(progress.currentStep),

nextStep: this.getNextStep(progress)

};

}

}

// src/components/onboarding/OnboardingFlowGuard.tsx

import React, { useEffect, useState } from 'react';

import { useNavigate, useLocation } from 'react-router-dom';

import { OnboardingFlowService } from '../../services/OnboardingFlowService';

import { NavigationContext } from '../../types/onboarding';

interface OnboardingFlowGuardProps {

children: React.ReactNode;

stepId: string;

userId: string;

}

export const OnboardingFlowGuard: React.FC<OnboardingFlowGuardProps> = ({

children,

stepId,

userId

}) => {

const [navigationContext, setNavigationContext] = useState<NavigationContext | null>(null);

const [isLoading, setIsLoading] = useState(true);

const navigate = useNavigate();

const location = useLocation();

const flowService = OnboardingFlowService.getInstance();

useEffect(() => {

checkAccess();

}, [stepId, userId, location.pathname]);

const checkAccess = async () => {

try {

setIsLoading(true);

const context = await flowService.getNavigationContext(userId, stepId);

setNavigationContext(context);

// If user can't access this step, redirect

if (!context.canAccess && context.redirectTo) {

navigate(context.redirectTo, { replace: true });

return;

}

} catch (error) {

console.error('Error checking step access:', error);

// On error, redirect to welcome

navigate('/onboarding/welcome', { replace: true });

} finally {

setIsLoading(false);

}

};

if (isLoading) {

return (

<div className="flex items-center justify-center min-h-screen">

<div className="animate-spin rounded-full h-12 w-12 border-b-2 border-blue-600"></div>

</div>

);

}

if (!navigationContext?.canAccess) {

return null; // Will redirect in useEffect

}

return <>{children}</>;

};

// src/components/onboarding/UniversalContinueButton.tsx

import React, { useState } from 'react';

import { useNavigate } from 'react-router-dom';

import { OnboardingFlowService } from '../../services/OnboardingFlowService';

interface UniversalContinueButtonProps {

userId: string;

currentStep: string;

completionData: Record<string, any>;

isValid: boolean;

className?: string;

children?: React.ReactNode;

}

export const UniversalContinueButton: React.FC<UniversalContinueButtonProps> = ({

userId,

currentStep,

completionData,

isValid,

className = '',

children = 'Continue'

}) => {

const [isLoading, setIsLoading] = useState(false);

const navigate = useNavigate();

const flowService = OnboardingFlowService.getInstance();

const handleContinue = async () => {

if (!isValid || isLoading) return;

try {

setIsLoading(true);

const result = await flowService.proceedToNextStep(userId, currentStep, completionData);

if (result.isComplete) {

// Show completion celebration or go to dashboard

navigate('/onboarding/complete');

} else {

navigate(result.route);

}

} catch (error) {

console.error('Error proceeding to next step:', error);

// Show error message to user

alert('Something went wrong. Please try again.');

} finally {

setIsLoading(false);

}

};

return (

<button

onClick={handleContinue}

disabled={!isValid || isLoading}

className={`

px-6 py-3 bg-blue-600 text-white rounded-lg font-medium

hover:bg-blue-700 disabled:bg-gray-300 disabled:cursor-not-allowed

transition-colors duration-200 flex items-center justify-center

${className}

`}

>

{isLoading ? (

<>

<div className="animate-spin rounded-full h-4 w-4 border-b-2 border-white mr-2"></div>

Processing...

</>

) : (

children

)}

</button>

);

};

// src/components/onboarding/OnboardingProgressBar.tsx

import React, { useEffect, useState } from 'react';

import { OnboardingFlowService } from '../../services/OnboardingFlowService';

import { OnboardingProgress } from '../../types/onboarding';

interface OnboardingProgressBarProps {

userId: string;

currentStep: string;

}

export const OnboardingProgressBar: React.FC<OnboardingProgressBarProps> = ({

userId,

currentStep

}) => {

const [progress, setProgress] = useState<OnboardingProgress | null>(null);

const [summary, setSummary] = useState<any>(null);

const flowService = OnboardingFlowService.getInstance();

useEffect(() => {

loadProgress();

}, [userId, currentStep]);

const loadProgress = async () => {

try {

const progressData = await flowService.fetchProgress(userId);

setProgress(progressData);

setSummary(flowService.getProgressSummary(progressData));

} catch (error) {

console.error('Error loading progress:', error);

}

};

if (!progress || !summary) {

return null;

}

const steps = flowService.getAllSteps().filter(s => s.required);

return (

<div className="w-full max-w-4xl mx-auto px-4 py-6">

{/\* Progress Bar \*/}

<div className="mb-4">

<div className="flex justify-between items-center mb-2">

<span className="text-sm font-medium text-gray-700">

Step {summary.completedSteps + 1} of {summary.totalSteps}

</span>

<span className="text-sm font-medium text-gray-700">

{progress.progressPercentage}% 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: `${progress.progressPercentage}%` }}

></div>

</div>

</div>

{/\* Step Indicators \*/}

<div className="flex justify-between items-center">

{steps.map((step, index) => {

const isCompleted = progress.completedSteps.includes(step.id);

const isCurrent = step.id === currentStep;

const isAccessible = flowService.canAccessStep(step.id, progress);

return (

<div key={step.id} className="flex flex-col items-center">

<div className={`

w-8 h-8 rounded-full flex items-center justify-center text-sm font-medium

${isCompleted ? 'bg-green-500 text-white' :

isCurrent ? 'bg-blue-600 text-white' :

isAccessible ? 'bg-gray-300 text-gray-600' : 'bg-gray-200 text-gray-400'}

`}>

{isCompleted ? '✓' : index + 1}

</div>

<span className={`

text-xs mt-1 text-center max-w-16

${isCurrent ? 'text-blue-600 font-medium' : 'text-gray-500'}

`}>

{step.title.split(' ')[0]}

</span>

</div>

);

})}

</div>

{/\* Time Estimate \*/}

{summary.estimatedTimeRemaining > 0 && (

<div className="text-center mt-4">

<span className="text-sm text-gray-600">

About {summary.estimatedTimeRemaining} minutes remaining

</span>

</div>

)}

</div>

);

};

// Example: Updated Onboarding Step Component

// src/components/onboarding/steps/WelcomeStep.tsx

import React, { useState, useEffect } from 'react';

import { OnboardingFlowGuard } from '../OnboardingFlowGuard';

import { UniversalContinueButton } from '../UniversalContinueButton';

import { OnboardingProgressBar } from '../OnboardingProgressBar';

interface WelcomeStepProps {

userId: string;

}

export const WelcomeStep: React.FC<WelcomeStepProps> = ({ userId }) => {

const [hasViewedWelcome, setHasViewedWelcome] = useState(false);

const [startTime] = useState(Date.now());

useEffect(() => {

// Mark as viewed after 3 seconds

const timer = setTimeout(() => {

setHasViewedWelcome(true);

}, 3000);

return () => clearTimeout(timer);

}, []);

const getCompletionData = () => {

return {

viewed\_welcome\_screen: hasViewedWelcome,

clicked\_get\_started: true,

time\_spent: Math.round((Date.now() - startTime) / 60000) // minutes

};

};

return (

<OnboardingFlowGuard stepId="welcome" userId={userId}>

<div className="min-h-screen bg-gradient-to-br from-blue-50 to-indigo-100">

<OnboardingProgressBar userId={userId} currentStep="welcome" />

<div className="container mx-auto px-4 py-12">

<div className="max-w-2xl mx-auto text-center">

<div className="mb-8">

<h1 className="text-4xl font-bold text-gray-900 mb-4">

Welcome to Mingus! 👋

</h1>

<p className="text-xl text-gray-600 mb-6">

Finally, a finance app that gets your real life

</p>

</div>

<div className="bg-white rounded-lg shadow-lg p-8 mb-8">

<h2 className="text-2xl font-semibold mb-6">What makes Mingus different?</h2>

<div className="grid md:grid-cols-3 gap-6 mb-8">

<div className="text-center">

<div className="text-3xl mb-2">🏃‍♀️</div>

<h3 className="font-medium mb-2">Health Connection</h3>

<p className="text-sm text-gray-600">

See how stress and wellness impact your spending

</p>

</div>

<div className="text-center">

<div className="text-3xl mb-2">📊</div>

<h3 className="font-medium mb-2">Job Security</h3>

<p className="text-sm text-gray-600">

Early warning system for employment risks

</p>

</div>

<div className="text-center">

<div className="text-3xl mb-2">🎯</div>

<h3 className="font-medium mb-2">Life Planning</h3>

<p className="text-sm text-gray-600">

Plan for what matters: birthdays, trips, goals

</p>

</div>

</div>

<div className="bg-blue-50 rounded-lg p-4 mb-6">

<p className="text-sm text-blue-800">

⏱️ This setup takes about 25 minutes. You can save your progress and return anytime.

</p>

</div>

</div>

<div className="flex justify-center">

<UniversalContinueButton

userId={userId}

currentStep="welcome"

completionData={getCompletionData()}

isValid={hasViewedWelcome}

className="text-lg px-8 py-4"

>

Get Started →

</UniversalContinueButton>

</div>

</div>

</div>

</div>

</OnboardingFlowGuard>

);

};

// Example Route Setup with Guards

// src/App.tsx (or your routing file)

import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';

import { WelcomeStep } from './components/onboarding/steps/WelcomeStep';

// ... other step imports

function App() {

const userId = "user\_123"; // Get from auth context

return (

<Router>

<Routes>

{/\* Onboarding Routes \*/}

<Route path="/onboarding/welcome" element={<WelcomeStep userId={userId} />} />

<Route path="/onboarding/profile" element={<ProfileSetupStep userId={userId} />} />

<Route path="/onboarding/financial" element={<FinancialProfileStep userId={userId} />} />

<Route path="/onboarding/goals" element={<GoalsSetupStep userId={userId} />} />

<Route path="/onboarding/health" element={<HealthAssessmentStep userId={userId} />} />

<Route path="/onboarding/data-input" element={<DataInputStep userId={userId} />} />

<Route path="/onboarding/preferences" element={<PreferencesStep userId={userId} />} />

<Route path="/onboarding/dashboard-intro" element={<DashboardIntroStep userId={userId} />} />

<Route path="/onboarding/complete" element={<OnboardingComplete userId={userId} />} />

{/\* Other routes \*/}

<Route path="/dashboard" element={<Dashboard />} />

</Routes>

</Router>

);

}