

# 금융 포트폴리오 플랫폼: 온보딩 기능 구현 가이드

## 1. 프로젝트 초기화

### Step 1: React+Vite+TypeScript 프로젝트 생성

```
bash
npm create vite@latest portfolio-platform -- --template react-ts
cd portfolio-platform
npm install
```

### Step 2: 필수 라이브러리 설치

```
bash

# UI 및 상태관리
npm install zustand react-router-dom axios

# 폼 관리 및 유효성 검사
npm install react-hook-form zod @hookform/resolvers

# UI 컴포넌트 (선택: 기본 스타일)
npm install clsx

# 개발 도구
npm install -D tailwindcss postcss autoprefixer
npx tailwindcss init -p
```

### Step 3: 폴더 구조 생성

```
bash
mkdir -p src/{api,store,pages,components/{onboarding,common},hooks,types,utils,styles}
```

## 2. 프로젝트 구조

```
portfolio-platform/
├── src/
│   ├── types/
│   │   └── onboarding.ts      # 온보딩 타입 정의
│   ├──
│   ├── api/
│   │   └── onboardingClient.ts # API 호출 로직
```

```
| |
| |   └── store/
| |     └── onboardingStore.ts    # Zustand 상태관리
| |
| |   └── pages/
| |     └── OnboardingPage.tsx    # 온보딩 페이지
| |
| |   └── components/
| |     └── onboarding/
| |       ├── OnboardingForm.tsx  # 메인 폼 컴포넌트
| |       ├── QuestionSection.tsx # 질문 섹션
| |       └── ProgressBar.tsx     # 진행률 표시
| |
| |     └── common/
| |       ├── Button.tsx
| |       ├── Input.tsx
| |       └── Select.tsx
| |
| |   └── utils/
| |     ├── validation.ts         # 유효성 검사
| |     └── constants.ts         # 상수 (선택지 등)
| |
| |   └── App.tsx
| |   └── main.tsx
| |   └── index.css               # Tailwind CSS 임포트
| |
| └── vite.config.ts
| └── tsconfig.json
| └── package.json
```

### 3. 타입 정의 (src/types/onboarding.ts)

온보딩 데이터 구조를 정의합니다. 이 부분을 가장 먼저 정의해서 추후 업데이트 시 여기만 수정하면 됩니다.

typescript

```
// src/types/onboarding.ts
```

```
export interface BusinessField {  
  category: 'finance' | 'marketing'  
  subcategories: string[]  
}
```

```
export interface OnboardingData {  
  // Question 1: 사업 분야  
  businessField?: 'finance' | 'marketing'  
  businessSubfield?: string  
  
  // Question 2: 자산 수  
  assetCount?: string  
  
  // Question 3-4: 파생상품  
  hasDerivatives?: boolean  
  includeDerivativesInOptimization?: boolean  
  
  // Question 5: 거래 비용  
  includeTransactionCost?: boolean  
  
  // Question 6: 최적화 수행 기간  
  optimizationPeriod?: 'single' | 'multiple'  
  
  // Question 7: 제약 조건 (복수선택)  
  constraints?: string[]  
  
  // Question 8: 규제 제약  
  hasRegulationConstraints?: boolean  
  regulationDetails?: string  
  
  // Question 9: 최적화 프레임워크  
  optimizationFramework?: string  
  
  // Question 10: 포트폴리오 관리 전략  
  hasPreferredStrategy?: boolean  
  preferredStrategy?: string  
  strategyRankingCriteria?: string  
  
  // Question 11: 신규 자산 추가  
  addNewAssets?: boolean  
  newAssets?: string  
  
  // Question 12: 서비스 플랜  
  servicePlan?: string
```

```
// Question 13: 리밸런싱
rebalancing?: boolean
rebalancingType?: 'auto' | 'manual' | 'semi-auto'
```

```
// Question 14: 포트폴리오 건전성 확인
checkPortfolioHealth?: boolean
```

```
// Question 15: 요금 방식
pricingModel?: string
```

```
// Question 16-19: 시스템 프로세스
dataSubmitted?: boolean
optimizationStarted?: boolean
resultsReceived?: boolean
rebalancingStarted?: boolean
```

```
// 메타데이터
createdAt?: string
updatedAt?: string
currentStep?: number
}
```

```
// 진행 상태
export type OnboardingStep =
  | 'pending'
  | 'in-progress'
  | 'completed'
  | 'error'

export interface OnboardingStatus {
  status: OnboardingStep
  completedSteps: number
  totalSteps: number
  lastUpdated: string
}
```

---

## 4. 상수 정의 (src/utils/constants.ts)

모든 선택지를 한 곳에서 관리합니다. 업데이트 시 여기만 수정하면 UI에 반영됩니다.

```
typescript
```

```
// src/utils/constants.ts
```

```
export const ONBOARDING_QUESTIONS = {  
  businessField: {  
    label: '사업 분야를 알려주세요',  
    options: [  
      {  
        value: 'finance',  
        label: '금융',  
        subcategories: ['은행', '증권', '자산운용', '기타']  
      },  
      {  
        value: 'marketing',  
        label: '마케팅',  
        subcategories: ['온라인', '오프라인']  
      }  
    ]  
  },  
  
  assetCount: {  
    label: '현재 포트폴리오에 보유 중인 자산은 몇 개입니까?',  
    options: ['50개 미만', '50개~200개', '200개~2,000개', '2,000개 초과']  
  },  
  
  hasDerivatives: {  
    label: '포트폴리오에 파생상품이 포함되어 있습니까?',  
  },  
  
  includeDerivatives: {  
    label: '포트폴리오 최적화에 파생상품을 포함시키시겠습니까?',  
  },  
  
  includeTransactionCost: {  
    label: '포트폴리오 최적화에 거래 비용을 포함하시겠습니까?',  
  },  
  
  optimizationPeriod: {  
    label: '포트폴리오 최적화 수행 기간을 선택해주세요',  
    options: ['단일 기간', '다중 기간(예: K개월 동안 매일 최적화)']  
  },  
  
  constraints: {  
    label: '고려하고 싶은 제약 조건을 모두 선택해주세요 (복수선택)',  
    options: [  
      '양(+)의 값 (No Short)',  
      '예산',  
    ]  
  }  
}
```

```
'자기자본',
'보유량',
'종목 수',
'레버리지',
'회전율',
'시장 중립',
'최대 포지션',
'희소성'
]
},

hasRegulationConstraints: {
  label: '포트폴리오 관리에 적용되는 규제 제약이 있습니까?',
},

regulationDetails: {
  label: '구체적인 규제 내용을 알려주세요',
},

optimizationFramework: {
  label: '선호하는 최적화 프레임워크를 선택해주세요',
  options: [
    '스토캐스틱 포트폴리오 최적화(Stochastic Portfolio Optimization, SPO)',
    '팩터 기반 로버스트 최적화(Factor Based Robust Optimization, FBRO)',
    '강화학습 기반 최적화(Reinforcement Learning Based Optimization, RLBO)',
    '평균-분산 최적화(Mean-Variance Optimization, MVO)',
    '글로벌 최소 분산 포트폴리오 최적화(Global Minimum Variance Portfolio Optimization, GMVPO)',
    '샤프 지수 최대화 포트폴리오 최적화(Maximum Sharpe Ratio Portfolio Optimization, MSRPO)',
    '효용 함수 최적화: 로그 | 제공근 | 역수 | 거듭 제곱 | 지수',
    '평균-CVaR 최적화'
  ]
},

hasPreferredStrategy: {
  label: '선호하는 포트폴리오 관리 전략이 있습니까?',
},

preferredStrategy: {
  label: '어떤 전략을 선호하십니까?',
  options: [
    '매수 후 보유(Buy & Hold)',
    '동일 가중치 포트폴리오(1/N)',
    '퀀타일 포트폴리오(Quintile)',
    '전역 최대 수익 포트폴리오'
  ]
},
```

```
strategyRankingCriteria: {
  label: '순위 산정 시 선호하는 기준을 기입해주세요',
},

addNewAssets: {
  label: '최적화를 위해 현재 포트폴리오에 포함되지 않은 새로운 자산을 추가하시겠습니까?',
},

newAssets: {
  label: '추가할 자산을 알려주세요',
},

servicePlan: {
  label: '이용 가능한 서비스 플랜',
  options: [
    '부분 양자 기술, 부분 고전 기술 사용(예: 양자 자산 선택, 고전 최적화)',
    '완전 양자 컴퓨팅 사용',
    '전처리 기술 포함(PCA 통한 그룹화 등)'
  ]
},

rebalancing: {
  label: '최적화 결과에 따라 포트폴리오를 리밸런싱하시겠습니까?',
},

rebalancingType: {
  label: '리밸런싱을 자동으로 진행하시겠습니까, 아니면 수동으로 진행하시겠습니까?',
  options: ['자동', '수동', '반자동']
},

checkPortfolioHealth: {
  label: '포트폴리오의 건전성을 확인하시겠습니까?',
},

pricingModel: {
  label: '요금 안내',
  options: [
    '계층적 그룹 최적화(상위 계층 고전 최적화 | 하위 계층 양자 최적화)',
    '계층적 그룹 최적화(양자 최적화)',
    '계층적 그룹 최적화(고전 최적화)',
    '일괄 최적화(양자 최적화)'
  ]
},

agreedToTerms: {
  label: '가격 및 기타 약관에 동의하십니까?',
}
```

```
}  
} as const
```

## 5. 상태관리 (src/store/onboardingStore.ts)

Zustand로 전체 온보딩 데이터 관리합니다.

typescript



```
// src/store/onboardingStore.ts
```

```
import { create } from 'zustand'
import { OnboardingData, OnboardingStatus } from '@types/onboarding'
```

```
interface OnboardingStore {
  formData: OnboardingData
  status: OnboardingStatus
```

```
// 액션
```

```
  updateFormData: (data: Partial<OnboardingData>) => void
  setCurrentStep: (step: number) => void
  resetForm: () => void
  submitForm: () => Promise<void>
}
```

```
const initialFormData: OnboardingData = {
  currentStep: 1
}
```

```
const initialStatus: OnboardingStatus = {
  status: 'pending',
  completedSteps: 0,
  totalSteps: 19,
  lastUpdated: new Date().toISOString()
}
```

```
export const useOnboardingStore = create<OnboardingStore>((set, get) => ({
  formData: initialFormData,
  status: initialStatus,

  updateFormData: (data) =>
    set((state) => ({
      formData: { ...state.formData, ...data, updatedAt: new Date().toISOString() }
    })),

  setCurrentStep: (step) =>
    set((state) => ({
      formData: { ...state.formData, currentStep: step }
    })),

  resetForm: () =>
    set({
      formData: initialFormData,
      status: initialStatus
    }),
})
```

```
submitForm: async () => {
  const { formData } = get()
  set((state) => ({ status: { ...state.status, status: 'in-progress' } })))

  try {
    // 백엔드 호출 (나중에 구현)
    // await onboardingAPI.submit(formData)

    set((state) => ({
      status: {
        ...state.status,
        status: 'completed',
        completedSteps: state.status.totalSteps,
        lastUpdated: new Date().toISOString()
      }
    })))
  } catch (error) {
    set((state) => ({ status: { ...state.status, status: 'error' } })))
    throw error
  }
}
```

## 6. 공통 UI 컴포넌트

### Button 컴포넌트

typescript

```
// src/components/common/Button.tsx
```

```
interface ButtonProps extends React.ButtonHTMLAttributes<HTMLButtonElement> {  
  variant?: 'primary' | 'secondary' | 'outline'  
  size?: 'sm' | 'md' | 'lg'  
  isLoading?: boolean  
}  
  
export function Button({  
  variant = 'primary',  
  size = 'md',  
  isLoading = false,  
  children,  
  disabled,  
  ...props  
}: ButtonProps) {  
  const baseStyle =  
    'font-medium rounded-lg transition-colors focus:outline-none disabled:opacity-50'  
  
  const variants = {  
    primary: 'bg-blue-600 text-white hover:bg-blue-700',  
    secondary: 'bg-gray-200 text-gray-900 hover:bg-gray-300',  
    outline: 'border-2 border-blue-600 text-blue-600 hover:bg-blue-50'  
  }  
  
  const sizes = {  
    sm: 'px-3 py-1.5 text-sm',  
    md: 'px-4 py-2 text-base',  
    lg: 'px-6 py-3 text-lg'  
  }  
  
  return (  
    <button  
      disabled={disabled || isLoading}  
      className={` ${baseStyle} ${variants[variant]} ${sizes[size]} `}  
      {...props}  
    >  
      {isLoading ? '처리 중...' : children}  
    </button>  
  )  
}
```

## Select 컴포넌트 (라디오 버튼 + 셀렉트)

typescript

```
// src/components/common/Select.tsx
```

```
interface SelectProps {  
  label?: string  
  options: string[]  
  value?: string  
  onChange: (value: string) => void  
  multiple?: boolean  
  type?: 'radio' | 'checkbox' | 'select'  
}
```

```
export function Select({  
  label,  
  options,  
  value,  
  onChange,  
  multiple = false,  
  type = 'radio'  
}: SelectProps) {  
  return (  
    <div className="space-y-2">  
      {label && <label className="block font-medium text-gray-700">{label}</label>}  
  
      {type === 'checkbox' && (  
        <div className="space-y-2">  
          {options.map((option) => (  
            <label key={option} className="flex items-center">  
              <input  
                type="checkbox"  
                className="mr-2"  
                defaultChecked={value?.includes(option)}  
                onChange={(e) => {  
                  // 복수선택 로직  
                }}  
              />  
              <span>{option}</span>  
            </label>  
          ))}  
        </div>  
      )}  
  
      {type === 'radio' && (  
        <div className="space-y-2">  
          {options.map((option) => (  
            <label key={option} className="flex items-center">  
              <input
```

```

      type="radio"
      name={label}
      value={option}
      checked={value === option}
      onChange={(e) => onChange(e.target.value)}
      className="mr-2"
    />
    <span>{option}</span>
  </label>
)}}
</div>
)}

{type === 'select' && (
  <select
    value={value || ''}
    onChange={(e) => onChange(e.target.value)}
    className="w-full border border-gray-300 rounded-lg px-3 py-2"
  >
    <option value="">선택해주세요</option>
    {options.map((option) => (
      <option key={option} value={option}>
        {option}
      </option>
    ))}
  </select>
)}
</div>
)
}

```

## 7. 온보딩 폼 컴포넌트 (단계별)

typescript

```
// src/components/onboarding/OnboardingForm.tsx
```

```
import { useState } from 'react'
import { useOnboardingStore } from '@store/onboardingStore'
import { ONBOARDING_QUESTIONS } from '@utils/constants'
import { Button } from '@components/common/Button'
import { Select } from '@components/common/Select'

export function OnboardingForm() {
  const { formData, updateFormData, setCurrentStep, submitForm } = useOnboardingStore()
  const currentStep = formData.currentStep || 1
  const totalSteps = 19

  const handleNext = () => {
    if (currentStep < totalSteps) {
      setCurrentStep(currentStep + 1)
    }
  }

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

  const handleSubmit = async () => {
    try {
      await submitForm()
      alert('온보딩 완료!')
    } catch (error) {
      alert('오류가 발생했습니다')
    }
  }

  return (
    <div className="max-w-2xl mx-auto p-6">
      { /* 진행률 표시 */ }
      <div className="mb-6">
        <div className="flex justify-between text-sm text-gray-600 mb-2">
          <span>Step {currentStep} of {totalSteps}</span>
          <span>{Math.round((currentStep / totalSteps) * 100)}%</span>
        </div>
        <div className="w-full bg-gray-200 rounded-full h-2">
          <div
            className="bg-blue-600 h-2 rounded-full transition-all"
            style={{ width: `${(currentStep / totalSteps) * 100}%` }}
          </div>
        </div>
      </div>
    </div>
  )
}
```

```

    />
  </div>
</div>

{/* 질문에 따른 렌더링 */}
<div className="bg-white p-6 rounded-lg border border-gray-200 mb-6">
  <QuestionRenderer currentStep={currentStep} formData={formData} updateFormData={updateFormDa
</div>

{/* 네비게이션 버튼 */}
<div className="flex justify-between">
  <Button
    variant="outline"
    onClick={handlePrevious}
    disabled={currentStep === 1}
  >
    이전
  </Button>

  {currentStep < totalSteps ? (
    <Button variant="primary" onClick={handleNext}>
      다음
    </Button>
  ) : (
    <Button variant="primary" onClick={handleSubmit}>
      완료
    </Button>
  )}
</div>
</div>
)
}

// 각 단계별 질문 렌더러
function QuestionRenderer({ currentStep, formData, updateFormData }: any) {
  switch (currentStep) {
    case 1:
      return (
        <div>
          <h2 className="text-lg font-bold mb-4">
            {ONBOARDING_QUESTIONS.businessField.label}
          </h2>
          <Select
            type="radio"
            options={ONBOARDING_QUESTIONS.businessField.options.map(o => o.label)}
            value={formData.businessField}
            onChange={(value) => updateFormData({ businessField: value })}

```

```

    />
  </div>
)

case 2:
  return (
    <div>
      <h2 className="text-lg font-bold mb-4">
        {ONBOARDING_QUESTIONS.assetCount.label}
      </h2>
      <Select
        type="radio"
        options={ONBOARDING_QUESTIONS.assetCount.options}
        value={formData.assetCount}
        onChange={(value) => updateFormData({ assetCount: value })}
      />
    </div>
  )

case 3:
  return (
    <div>
      <h2 className="text-lg font-bold mb-4">
        {ONBOARDING_QUESTIONS.hasDerivatives.label}
      </h2>
      <Select
        type="radio"
        options={['예', '아니요']}
        value={formData.hasDerivatives ? '예' : '아니요'}
        onChange={(value) => updateFormData({ hasDerivatives: value === '예' })}
      />
    </div>
  )

// ... case 4-19 추가 (코드는 유사함)

default:
  return null
}
}

```

## 8. API 클라이언트 (향후 백엔드 연동용)

typescript



```
// src/api/onboardingClient.ts
```

```
import axios from 'axios'
import { OnboardingData } from '@types/onboarding'

const API_URL = process.env.REACT_APP_API_URL || 'http://localhost:8000'

export const onboardingAPI = {
  submit: async (data: OnboardingData) => {
    const response = await axios.post(`${API_URL}/api/onboarding/submit`, data)
    return response.data
  },

  getStatus: async (userId: string) => {
    const response = await axios.get(`${API_URL}/api/onboarding/status/${userId}`)
    return response.data
  }
}
```

## 9. 다음 단계

### 아직 안 된 것들

1. ☒ 타입 정의
2. ☒ 상수 관리
3. ☒ 상태관리 구조
4. ☒ 기본 UI 컴포넌트
5. ☒ 질문 렌더러 (case 1-3만 예시)
6. ☒ case 4-19 완성 ← 당신이 할 부분
7. ☒ 유효성 검사 (react-hook-form 연동)
8. ☒ API 연동 (백엔드 준비 후)
9. ☒ 페이지 라우팅 (메인 페이지와 연동)

### 업데이트 시 수정할 부분

문서에서 온보딩 질문 변경

↓

src/utlis/constants.ts 수정

↓

src/types/onboarding.ts 수정 (필요 시)



QuestionRenderer에 새로운 case 추가



UI 자동 반영 