Step 4: TypeScript Interfaces Creation - Task Breakdown

Based on your existing code analysis, here are the step-by-step tasks to create comprehensive TypeScript interfaces:

Task 4.1: Create Core Type Definitions (1 hour)

File: `src/types/index.ts`

Purpose: Central type definitions file

```typescript
```

```typescript
// ===== CORE ENTITY INTERFACES =====

export interface User {
  id: number;
  name: string;
  email: string;
  createdAt?: string;
  updatedAt?: string;
  role?: 'admin' | 'manager' | 'member';
  profilePicture?: string;
}

export interface Team {
  id: number;
  name: string;
  status: 'active' | 'inactive' | 'archived';
  description: string;
  createdAt?: string;
  updatedAt?: string;
  memberCount?: number;
  ownerId?: number;
}

export interface Notification {
  id: number;
  message: string;
  type: 'success' | 'error' | 'warning' | 'info';
  duration: number;
  createdAt?: string;
  isRead?: boolean;
}

// ===== API RESPONSE INTERFACES =====

export interface ApiResponse<T = any> {
  success: boolean;
  data?: T;
  error?: string;
  message?: string;
}

export interface LoginResponse {
  access_token: string;
  user: User;
  expires_in?: number;
}
```

```typescript
export interface DashboardData {
  totalTeams: number;
  totalAssessments: number;
  avgCompatibility: number;
  predictedVelocity: number;
}

// ===== FORM DATA INTERFACES =====

export interface LoginFormData {
  email: string;
  password: string;
}

export interface RegisterFormData {
  name: string;
  email: string;
  password: string;
  confirmPassword: string;
}

export interface TeamFormData {
  name: string;
  description: string;
  status?: 'active' | 'inactive';
}
```

Task 4.2: Create Context Type Definitions (45 minutes)

File: `src/types/contexts.ts`

Purpose: Context-specific type definitions

```
typescript
```

```typescript
import { User, Team, Notification, ApiResponse, LoginFormData, RegisterFormData, TeamFormData } from './i

// ===== AUTH CONTEXT TYPES =====

export interface AuthContextType {
  user: User | null;
  isLoading: boolean;
  login: (email: string, password: string) => Promise<ApiResponse>;
  register: (userData: RegisterFormData) => Promise<ApiResponse>;
  logout: () => void;
}

// ===== NOTIFICATION CONTEXT TYPES =====

export interface NotificationContextType {
  notifications: Notification[];
  showNotification: (message: string, type?: Notification['type'], duration?: number) => void;
  removeNotification: (id: number) => void;
}

// ===== TEAM CONTEXT TYPES =====

export interface TeamContextType {
  teams: Team[];
  currentTeam: Team | null;
  loading: boolean;
  fetchTeams: () => Promise<void>;
  createTeam: (teamData: TeamFormData) => Promise<ApiResponse<Team>>;
  updateTeam: (teamId: number, updateData: Partial<Team>) => Promise<ApiResponse<Team>>;
  deleteTeam: (teamId: number) => Promise<ApiResponse>;
  selectTeam: (team: Team) => void;
}
```

Task 4.3: Create Component Prop Interfaces (30 minutes)

File: `src/types/components.ts`

Purpose: Component prop type definitions

```
typescript
```

```typescript
import { ReactNode, ButtonHTMLAttributes } from 'react';
import { Team } from './index';

// ===== COMPONENT PROP INTERFACES =====

export interface ButtonProps extends ButtonHTMLAttributes<HTMLButtonElement> {
  children: ReactNode;
  onClick?: () => void;
  className?: string;
  disabled?: boolean;
  variant?: 'primary' | 'secondary' | 'danger';
}

export interface LoadingSpinnerProps {
  size?: 'small' | 'medium' | 'large';
  color?: string;
}

export interface HeaderProps {
  onMenuToggle: () => void;
}

export interface SidebarProps {
  isOpen: boolean;
  onToggle: () => void;
}

export interface MenuItem {
  name: string;
  path: string;
  icon: string;
}

// ===== PROVIDER PROPS =====

export interface AuthProviderProps {
  children: ReactNode;
}

export interface NotificationProviderProps {
  children: ReactNode;
}

export interface TeamProviderProps {
  children: ReactNode;
}
```

Task 4.4: Update Context Files with Types (1 hour)

File: `src/contexts/AuthContext.tsx`

Action: Convert existing AuthContext to TypeScript

```typescript
import React, { useState, useEffect, createContext, useContext, ReactNode } from 'react';
import { AuthContextType, AuthProviderProps } from '../types/contexts';
import { User, ApiResponse, RegisterFormData } from '../types';

const AuthContext = createContext<AuthContextType | undefined>(undefined);

export const useAuth = (): AuthContextType => {
  const context = useContext(AuthContext);
  if (!context) {
    throw new Error('useAuth must be used within an AuthProvider');
  }
  return context;
};

export const AuthProvider: React.FC<AuthProviderProps> = ({ children }) => {
  const [user, setUser] = useState<User | null>(null);
  const [isLoading, setIsLoading] = useState<boolean>(true);

  // ... rest of your existing logic with proper typing
};
```

Task 4.5: Update Component Files with Types (1.5 hours)

File: `src/components/Button.tsx`

```typescript
```

```typescript
import React from 'react';
import { ButtonProps } from '../types/components';

export const Button: React.FC<ButtonProps> = ({
  children,
  onClick,
  className = '',
  disabled = false,
  variant = 'primary',
  ...props
}) => {
  // Component implementation with proper typing
};
```

File: `src/components/LoadingSpinner.tsx`

```typescript
import React from 'react';
import { LoadingSpinnerProps } from '../types/components';

export const LoadingSpinner: React.FC<LoadingSpinnerProps> = ({
  size = 'medium',
  color = 'blue-600'
}) => {
  // Component implementation
};
```

Task 4.6: Create Utility Types (20 minutes)

File: `src/types/utils.ts`

Purpose: Utility types and enums

```typescript
```

```
// ===== ENUMS =====

export enum UserRole {
  ADMIN = 'admin',
  MANAGER = 'manager',
  MEMBER = 'member'
}

export enum TeamStatus {
  ACTIVE = 'active',
  INACTIVE = 'inactive',
  ARCHIVED = 'archived'
}

export enum NotificationType {
  SUCCESS = 'success',
  ERROR = 'error',
  WARNING = 'warning',
  INFO = 'info'
}

// ===== UTILITY TYPES =====

export type LoadingState = 'idle' | 'loading' | 'success' | 'error';

export type FormErrors<T> = {
  [K in keyof T]?: string;
};

// Generic form state
export interface FormState<T> {
  data: T;
  errors: FormErrors<T>;
  isSubmitting: boolean;
  isValid: boolean;
}
```

Task 4.7: Update Main App File (30 minutes)

File: `src/App.tsx`

Action: Add proper typing to your main App component

```
typescript
```

```
import React, { useState } from 'react';
import { BrowserRouter as Router, Routes, Route, Navigate } from 'react-router-dom';
import { AuthProvider } from './contexts/AuthContext';
// ... other imports with proper types
```

Checklist - Required Actions:

- [ ] Create `src/types/index.ts` with core interfaces
- [ ] Create `src/types/contexts.ts` with context types
- [ ] Create `src/types/components.ts` with component prop types
- [ ] Create `src/types/utils.ts` with utility types and enums
- [ ] Update `src/contexts/AuthContext.tsx` with TypeScript
- [ ] Update `src/contexts/NotificationContext.tsx` with TypeScript
- [ ] Update `src/contexts/TeamContext.tsx` with TypeScript
- [ ] Update `src/components/Button.tsx` with proper props typing
- [ ] Update `src/components/LoadingSpinner.tsx` with proper props typing
- [ ] Update `src/App.tsx` with TypeScript imports and typing
- [ ] Update `src/main.tsx` with proper typing

Directory Structure After Completion:

```
src/
 ├── types/
|   ├── index.ts        # Core entity interfaces
|   ├── contexts.ts      # Context type definitions
|   ├── components.ts    # Component prop interfaces
|   └── utils.ts        # Utility types and enums
 ├── contexts/
|   ├── AuthContext.tsx
|   ├── NotificationContext.tsx
|   └── TeamContext.tsx
 ├── components/
|   ├── Button.tsx
|   ├── LoadingSpinner.tsx
|   ├── Header.tsx
|   └── Sidebar.tsx
 ├── pages/
|   ├── Login.tsx
|   ├── Register.tsx
|   └── Dashboard.tsx
 ├── App.tsx
 └── main.tsx
```

Estimated Time Breakdown:

- **Core interfaces:** 1 hour
- **Context types:** 45 minutes
- **Component types:** 30 minutes
- **Context file updates:** 1 hour
- **Component file updates:** 1.5 hours
- **Utility types:** 20 minutes
- **Main app updates:** 30 minutes

Total Estimated Time: 5 hours 15 minutes

Validation Steps:

1. Run `npm run type-check` (if configured)
2. Check for TypeScript errors in IDE
3. Verify all imports resolve correctly
4. Test component prop validation
5. Ensure context types work properly

This completes the TypeScript interface creation phase of your Frontend Foundation task.