Architecture du Projet Portfolio - Atomic Design

Vue d'ensemble

Ce projet est un portfolio professionnel développé en React avec TypeScript, organisé selon les principes de l'**Atomic Design**. L'architecture est conçue pour être maintenable, évolutive et impressionnante pour les recruteurs.

Principes Architecturaux

Atomic Design

L'Atomic Design divise les composants UI en 5 niveaux hiérarchiques :

1. **Atoms** : Composants de base (boutons, inputs, icônes)
2. **Molecules** : Combinaisons d'atoms (formulaires simples, cartes)
3. **Organisms** : Sections complètes de l'UI (navbar, footer, listes)
4. **Templates** : Structures de page sans contenu réel
5. **Pages** : Vues finales avec données réelles

Bonnes Pratiques

- **Séparation des responsabilités** : Chaque composant a un rôle clair
- **Réutilisabilité** : Composants modulaires et configurables
- **Maintenabilité** : Code propre, bien structuré
- **Performance** : Optimisations React (memo, lazy loading)
- **Accessibilité** : Conformité WCAG
- **Responsive Design** : Mobile-first avec Tailwind CSS

Structure du Projet

PROF

```
src/
├── components/
│   ├── atoms/          # Composants de base
│   │   ├── Button.tsx   # Bouton réutilisable
│   │   └── ...
│   ├── molecules/      # Combinaisons d'atoms
│   │   ├── FormField.tsx # Champ de formulaire
│   │   └── ...
│   ├── organisms/      # Sections UI complexes
│   │   ├── Header.tsx   # Navigation principale
│   │   ├── Hero.tsx     # Section d'accueil
│   │   ├── About.tsx    # Section à propos
│   │   ├── Footer.tsx   # Pied de page
│   │   ├── Projects.tsx # Galerie de projets
│   │   ├── Navigation.tsx # Exemple de navigation
│   │   └── ...
```

```
│   └── templates/       # Structures de page
│       └── PortfolioTemplate.tsx
├── pages/               # Pages finales
│   ├── Home.tsx
│   ├── Dashboard.tsx
│   └── Upload.tsx
├── services/            # Communication API
│   └── uploadService.ts
├── state/               # Gestion d'état global
│   ├── index.ts
│   └── themeContext.tsx
├── utils/               # Helpers et utilitaires
│   ├── validation.ts
│   └── ...
├── uploads/             # Fichiers uploadés
└── ...
```

Exemples de Composants

Atom : Button

```
// src/components/atoms/Button.tsx
interface ButtonProps {
  children: React.ReactNode;
  onClick?: () => void;
  variant?: 'primary' | 'secondary' | 'outline';
  size?: 'sm' | 'md' | 'lg';
  disabled?: boolean;
}

const Button: React.FC<ButtonProps> = ({
  children,
  onClick,
  variant = 'primary',
  size = 'md',
  disabled = false,
  className = '',
}) => {
  // Implémentation avec Tailwind CSS
  const classes = `font-semibold rounded-lg transition-all duration-200
${variantClasses[variant]} ${sizeClasses[size]}`;
  return (
    <button className={classes} onClick={onClick} disabled={disabled}>
      {children}
    </button>
  );
};
```

PROF

Molecule : FormField

```
// src/components/molecules/FormField.tsx
interface FormFieldProps {
  label: string;
  value: string;
  onChange: (value: string) => void;
  error?: string;
  required?: boolean;
}

const FormField: React.FC<FormFieldProps> = ({
  label,
  value,
  onChange,
  error,
  required = false,
}) => {
  return (
    <div className="mb-4">
      <label className="block text-sm font-medium mb-2">
        {label}{required && <span className="text-red-500">*</span>}
      </label>
      <input
        type="text"
        value={value}
        onChange={(e) => onChange(e.target.value)}
        className="w-full px-3 py-2 border rounded-lg focus:ring-2
focus:ring-blue-500"
      />
      {error && <p className="mt-1 text-sm text-red-500">{error}</p>}
    </div>
  );
};
```

PROF

Organism : Navigation

```
// src/components/organisms/Navigation.tsx
import { Link } from 'react-router-dom';
import Button from '../atoms/Button';

const Navigation: React.FC = () => {
  const navItems = [
    { label: 'Home', href: '/' },
    { label: 'About', href: '/about' },
    { label: 'Projects', href: '/projects' },
    { label: 'Contact', href: '/contact' },
  ];

  return (
    <nav className="flex items-center space-x-4">
```

```
      {navItems.map((item) => (
        <Link key={item.href} to={item.href}>
          <Button variant="outline" size="sm">
            {item.label}
          </Button>
        </Link>
      ))}
    </nav>
  );
};
```

Gestion d'État

Context API pour l'état global

```
// src/state/themeContext.tsx
import React, { createContext, useContext, useState, useEffect } from
'react';

type Theme = 'light' | 'dark';

interface ThemeContextType {
  theme: Theme;
  toggleTheme: () => void;
}

const ThemeContext = createContext<ThemeContextType | undefined>
(undefined);

export const useTheme = () => {
  const context = useContext(ThemeContext);
  if (!context) throw new Error('useTheme must be used within
ThemeProvider');
  return context;
};

export const ThemeProvider: React.FC<{ children: ReactNode }> = ({
children }) => {
  const [theme, setTheme] = useState<Theme>('light');

  useEffect(() => {
    const savedTheme = localStorage.getItem('theme') as Theme;
    if (savedTheme) setTheme(savedTheme);
  }, []);

  const toggleTheme = () => setTheme(prev => prev === 'light' ? 'dark' :
'light');

  return (
    <ThemeContext.Provider value={{ theme, toggleTheme }}>
```

```
      {children}
    </ThemeContext.Provider>
  );
};
```

Services API

Exemple de service d'upload

```
// src/services/uploadService.ts
export const uploadFile = async (file: File): Promise<string> => {
  const formData = new FormData();
  formData.append('file', file);

  const response = await fetch('/api/upload', {
    method: 'POST',
    body: formData,
  });

  if (!response.ok) throw new Error('Upload failed');

  const data = await response.json();
  return data.url;
};
```

Bonnes Pratiques de Développement

1. Nommage des Composants

- PascalCase pour les noms de composants
- Préfixes descriptifs (Button, InputField, etc.)

PROF

2. Props et Types

- Utiliser TypeScript pour toutes les interfaces
- Props optionnelles avec ?
- Valeurs par défaut appropriées

3. Gestion d'État

- Context API pour l'état global
- useState pour l'état local
- useReducer pour la logique complexe

4. Performance

- React.memo pour éviter les re-renders inutiles
- useCallback et useMemo pour les fonctions et valeurs

- Lazy loading des composants

5. Tests

- Tests unitaires avec Jest
- Tests d'intégration pour les composants complexes
- Tests E2E avec Cypress

6. Accessibilité

- Attributs ARIA appropriés
- Navigation au clavier
- Contraste des couleurs suffisant

Technologies Utilisées

- **React 18** : Bibliothèque UI moderne
- **TypeScript** : Typage statique
- **Tailwind CSS** : Framework CSS utilitaire
- **React Router** : Navigation
- **Lucide React** : Icônes
- **Framer Motion** : Animations
- **Vite** : Outil de build rapide

Déploiement et CI/CD

- Build optimisé avec Vite
- Déploiement sur Vercel/Netlify
- CI/CD avec GitHub Actions
- Tests automatisés
- Monitoring des performances

PROF

Cette architecture garantit un code maintenable, évolutif et professionnel, idéal pour un portfolio impressionnant.