Frontend Implementation Guide

Ce document décrit les composants et pages frontend à implémenter pour compléter l'architecture multi-tenant, multi-company.

📋 Composants à Implémenter

1. Company Management Page (`/company/manage`)

Pour: Agency Admin

Fonctionnalités:
- Formulaire pour créer la company si elle n'existe pas encore
- Formulaire pour modifier les informations de la company
- Affichage en read-only pour Agency User

Hooks utilisés:

```
import { useMyCompany, useCreateMyCompany, useUpdateMyCompany, useHasCompany } from
"@/lib/hooks/useCompany";
```

Structure:

```
// app/company/manage/page.tsx
"use client";

import { useMyCompany, useCreateMyCompany, useUpdateMyCompany } from "@/lib/hooks/
useCompany";

export default function CompanyManagePage() {
  const { data: company, isLoading } = useMyCompany();
  const createMutation = useCreateMyCompany();
  const updateMutation = useUpdateMyCompany();

  if (isLoading) return <div>Loading...</div>;

  if (!company) {
    // Afficher le formulaire de création
    return <CreateCompanyForm onSubmit={(data) => createMutation.mutate(data)} />;
  }

  // Afficher le formulaire de modification
  return <UpdateCompanyForm company={company} onSubmit={(data) => updateMuta-
tion.mutate(data)} />;
}
```

---

2. Bank Account Management Page (`/company/bank`)

Pour: Agency Admin (edit), Agency User (read-only)

Fonctionnalités:
- Formulaire pour créer/modifier le bank account de la company
- Affichage en read-only pour Agency User
- Lien vers la page Company Management si pas de company

Hooks utilisés:

```
import { useMyCompanyBank, useSetMyCompanyBank, useHasBankAccount } from "@/lib/hooks/
useBankAccount";
import { useHasCompany } from "@/lib/hooks/useCompany";
```

Structure:

```
// app/company/bank/page.tsx
"use client";

import { useMyCompanyBank, useSetMyCompanyBank } from "@/lib/hooks/useBankAccount";
import { useHasCompany } from "@/lib/hooks/useCompany";

export default function BankAccountPage() {
  const { hasCompany, isLoading: companyLoading } = useHasCompany();
  const { data: bank, isLoading: bankLoading } = useMyCompanyBank();
  const setMutation = useSetMyCompanyBank();

  if (companyLoading || bankLoading) return <div>Loading...</div>;

  if (!hasCompany) {
    return <div>Please create your company first. <Link href="/company/manage">Go to C
ompany Management</Link></div>;
  }

  return (
    <BankAccountForm
      bank={bank}
      onSubmit={(data) => setMutation.mutate(data)}
    />
  );
}
```

---

3. Contract Creation Modal - Company Selection

Modifications nécessaires:
- Ajouter sélection du type de company (Tenant vs Agency)
- Utiliser `useTenantCompanies()` pour la liste des tenant companies
- Utiliser `useAgencyCompanies()` pour la liste des agency companies (Platform Admin)
- Afficher automatiquement la company de l'Agency Admin

Exemple:

```
import { useTenantCompanies, useAgencyCompanies, useMyCompany } from "@/lib/hooks/
useCompany";

function ContractCreationModal() {
  const { data: tenantCompanies } = useTenantCompanies();
  const { data: agencyCompanies } = useAgencyCompanies(); // Si Platform Admin
  const { data: myCompany } = useMyCompany(); // Si Agency Admin

  // Pour Platform Admin: permettre de sélectionner tenant company et agency company
  // Pour Agency Admin: tenant company sélectionnable, agency company = myCompany
(automatique)

  return (
    <form>
      <Select label="Tenant Company (Client)" options={tenantCompanies} />
      {isPlatformAdmin ? (
        <Select label="Agency Company (Service Provider)" options={agencyCompanies} />
      ) : (
        <div>Agency Company: {myCompany?.name}</div>
      )}
    </form>
  );
}
```

---

4. Contract Participant Display Component

Fonctionnalité: Afficher “Company Name (represented by User Name)” ou “User Name (Individual Contractor)”

Exemple:

```tsx
// components/contract/ParticipantDisplay.tsx

interface ParticipantDisplayProps {
  userId: string;
}

export function ParticipantDisplay({ userId }: ParticipantDisplayProps) {
  // Appeler l'endpoint tRPC qui utilise getParticipantDisplayName()
  const { data: participant } = trpc.contract.getParticipantDisplay.useQuery({
userId });

  if (!participant) return <div>Loading...</div>;

  if (participant.type === "company") {
    return (
      <div>
        <strong>{participant.name}</strong>
        <span className="text-muted"> (represented by {participant.representedBy})</sp
an>
      </div>
    );
  }

  return (
    <div>
      <strong>{participant.name}</strong>
      <span className="text-muted"> (Individual Contractor)</span>
    </div>
  );
}
```

---

5. MSA Creation Form - Auto-Approver Notice

Modification nécessaire: Afficher un message informatif au lieu d'un sélecteur d'approver

Avant:

```tsx
<Select label="Approver" options={platformAdmins} />
```

Après:

```tsx
<div className="alert alert-info">
  <InfoIcon />
  <span>A platform approver will be automatically assigned when you submit this MSA.</
span>
</div>
```

---

6. User List Component - Company Display

Modification nécessaire: Afficher la company de chaque user

Exemple:

```tsx
// components/users/UserList.tsx

function UserListItem({ user }) {
  return (
    <div className="user-item">
      <div className="user-name">{user.name}</div>
      <div className="user-email">{user.email}</div>
      {user.company && (
        <div className="user-company">
          <BuildingIcon />
          <span>{user.company.name}</span>
          <Badge>{user.company.type}</Badge>
        </div>
      )}
    </div>
  );
}
```

🎨 UI/UX Recommendations

Company Type Badges

```tsx
function CompanyTypeBadge({ type }: { type: "tenant" | "agency" }) {
  const config = {
    tenant: { label: "Client", color: "blue" },
    agency: { label: "Agency", color: "green" },
  };

  const { label, color } = config[type];

  return <Badge color={color}>{label}</Badge>;
}
```

Permission-Based Rendering

```tsx
import { usePermissions } from "@/lib/hooks/usePermissions";

function CompanyManagementSection() {
  const { hasPermission } = usePermissions();

  const canCreate = hasPermission("company.create.own");
  const canUpdate = hasPermission("company.update.own");
  const canOnlyRead = hasPermission("company.read.own") && !canUpdate;

  if (canOnlyRead) {
    return <CompanyDisplayReadOnly />;
  }

  if (!canCreate && !canUpdate) {
    return <NoAccess />;
  }

  return <CompanyManagementForm canCreate={canCreate} canUpdate={canUpdate} />;
}
```

🔗 Routes à Ajouter

Route	Component	Permissions Required	Description
`/company/manage`	CompanyManagePage	`company.create.own`, `company.update.own`	Gérer sa company
`/company/bank`	BankAccountPage	`bankAccount.create.own`, `bankAccount.update.own`	Gérer le bank account
`/company/users`	CompanyUsersPage	`user.list.ownCompany`	Voir les users de la company

🧪 Testing Scenarios

Test 1: Agency Admin Creates Company

1. Se connecter en tant qu'Agency Admin sans company
2. Aller sur `/company/manage`
3. Voir le formulaire de création
4. Remplir et soumettre
5. Vérifier que `company.type = "agency"`
6. Vérifier que l'interface passe en mode "update"

Test 2: Agency Admin Sets Bank Account

1. Se connecter en tant qu'Agency Admin avec company
2. Aller sur `/company/bank`
3. Remplir le formulaire
4. Soumettre
5. Vérifier que le bank account est créé et lié à la company

Test 3: Agency User Sees Read-Only

1. Se connecter en tant qu'Agency User
2. Aller sur `/company/manage`
3. Vérifier que l'interface est en read-only
4. Vérifier qu'aucun bouton "Edit" n'est visible

Test 4: Platform Admin Creates MSA

1. Se connecter en tant que Platform Admin
2. Créer un MSA
3. Sélectionner Tenant Company et Agency Company
4. Uploader le document principal

5. Vérifier qu'un approver est automatiquement assigné
6. Vérifier le statut passe à `pending_approval`

Test 5: Contract Participant Display

1. Créer un contrat avec un participant qui a une company
2. Créer un contrat avec un participant sans company
3. Vérifier l'affichage:
   - Avec company: "Company Name (represented by User Name)"
   - Sans company: "User Name (Individual Contractor)"

📦 Components Library Structure

Créer une structure organisée de composants réutilisables:

```
components/
├── company/
│   ├── CompanyForm.tsx
│   ├── CompanyDisplay.tsx
│   ├── CompanyTypeBadge.tsx
│   └── CompanySelector.tsx
├── bank/
│   ├── BankAccountForm.tsx
│   ├── BankAccountDisplay.tsx
│   └── BankAccountReadOnly.tsx
├── contract/
│   ├── ParticipantDisplay.tsx
│   ├── CompanySelection.tsx
│   ├── MSAApproverNotice.tsx
│   └── ContractTypeBadge.tsx
└── user/
    ├── UserListWithCompany.tsx
    ├── UserCompanyDisplay.tsx
    └── UserCard.tsx
```

🚀 Quick Start

1. **Copier les hooks**:
   - `lib/hooks/useCompany.ts`
   - `lib/hooks/useBankAccount.ts`

2. **Créer les pages**:
   - `app/company/manage/page.tsx`
   - `app/company/bank/page.tsx`

3. **Modifier les modals existants**:
   - Contract creation modal
   - MSA creation modal

4. **Ajouter les composants de display**:
   - ParticipantDisplay

- CompanyTypeBadge
- UserCompanyDisplay

5. **Tester les workflows**:
   - Agency Admin crée company
   - Agency Admin crée bank account
   - Platform Admin crée MSA (auto-approver)
   - Affichage des participants

📝 Notes Importantes

- **Permissions**: Toujours vérifier les permissions avant d'afficher les actions
- **Error Handling**: Gérer les cas où la company ou le bank account n'existe pas
- **Loading States**: Afficher des skeletons pendant le chargement
- **Toasts**: Afficher des messages de succès/erreur après les mutations
- **Validation**: Valider les formulaires côté client avant soumission

---

🔄 Cycle de Développement Recommandé

1. **Phase 1**: Créer les pages de base (Company Manage, Bank Account)
2. **Phase 2**: Ajouter les composants de display (ParticipantDisplay, etc.)
3. **Phase 3**: Modifier les modals existants (Contract creation)
4. **Phase 4**: Améliorer l'UX (badges, icons, animations)
5. **Phase 5**: Tests end-to-end et fixes de bugs

📞 Support

Pour toute question sur l'implémentation:
- Consulter `docs/RBAC_PERMISSIONS.md` pour les permissions
- Consulter `CHANGELOG.md` pour les changements backend
- Consulter les helpers dans `server/helpers/`

Version: 1.0
Date: 2025-11-26
Status: Ready for Implementation