VIVE-CREDIT - Task-uri Frontend

🔐 1. AUTENTIFICARE & ONBOARDING

Task 1.1: Pagini Login și Register

Timp estimat: 8-10 ore | **Dificultate:** 🟢 Mediu-Ușor

Ce vei construi: Două pagini simple de autentificare - una pentru login și una pentru înregistrare.

Cerințe (doar acestea!):

- [ ] Pagină Login cu: email, parolă, buton "Intră"
- [ ] Pagină Register cu: email, parolă, confirmă parola, buton "Creează cont"
- [ ] Validare pe email (să fie format valid) și parolă (minim 8 caractere)
- [ ] Mesaje de eroare clare când validarea eșuează
- [ ] Design responsive (să arate bine pe mobil)

Cum să abordezi (pas cu pas):

Pasul 1: Creează structura fișierelor
└── src/modules/auth/pages/
    ├── LoginPage.tsx
    └── RegisterPage.tsx

Pasul 2: Începe cu LoginPage
- Creează un form simplu cu 2 inputuri
- Folosește useState pentru email și password
- Adaugă un buton de submit

Pasul 3: Adaugă validarea
- Creează funcții simple de validare
- Afișează erori sub fiecare input

Pasul 4: Copiază structura pentru RegisterPage
- Adaugă câmpul extra pentru confirm password

Pasul 5: Styling cu Tailwind
- Centrează form-ul pe pagină
- Adaugă spațiere și culori

Cod starter pentru validare:

```
// Validare email simplă
const isValidEmail = (email: string): boolean => {
  return /^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(email);
};

// Validare parolă
const isValidPassword = (password: string): boolean => {
  return password.length >= 8;
};
```

Resurse utile:

- React Hook Form - Get Started
- Tailwind Forms

Task 1.2: Wizard Completare Profil (Simplificat)

Timp estimat: 10-12 ore | **Dificultate:** 🟡 Mediu

Ce vei construi: Un formular în 3 pași pentru completarea profilului utilizatorului.

Cerințe (reduse de la 4 la 3 steps):

- [ ] Step 1: Informații personale (nume, prenume, telefon)
- [ ] Step 2: Adresă (oraș, stradă, cod poștal)
- [ ] Step 3: Rezumat și confirmare
- [ ] Indicator vizual care arată la ce pas ești (1/3, 2/3, 3/3)
- [ ] Butoane "Înapoi" și "Continuă"
- [ ] Validare la fiecare pas înainte de a merge mai departe

Cum să abordezi:

Structura componentei principale:

```
function OnboardingWizard() {
  const [currentStep, setCurrentStep] = useState(1);
  const [formData, setFormData] = useState({
    // toate datele din toate stepurile
  });

  const nextStep = () => setCurrentStep(prev => prev + 1);
  const prevStep = () => setCurrentStep(prev => prev - 1);

  return (
    <div>
      <StepIndicator current={currentStep} total={3} />
```

```
      {currentStep === 1 && <PersonalInfoStep />}
      {currentStep === 2 && <AddressStep />}
      {currentStep === 3 && <SummaryStep />}

      <NavigationButtons />
    </div>
  );
}
```

Hint pentru StepIndicator:

```
function StepIndicator({ current, total }: { current: number; total: number }) {
  return (
    <div className="flex gap-2">
      {Array.from({ length: total }, (_, i) => (
        <div
          key={i}
          className={`w-8 h-8 rounded-full flex items-center justify-center
            ${i + 1 <= current ? 'bg-blue-500 text-white' : 'bg-gray-200'}`}
        >
          {i + 1}
        </div>
      ))}
    </div>
  );
}
```

Task 1.3: Pagină "Am uitat parola"

Timp estimat: 4-6 ore | **Dificultate:** 🟢 Ușor

Ce vei construi: O pagină simplă unde userul introduce email-ul pentru a primi link de resetare.

Cerințe:

- [ ] Input pentru email
- [ ] Buton "Trimite link"
- [ ] După submit: afișează mesaj de succes ("Verifică email-ul")
- [ ] Link înapoi la Login

Cod complet de referință:

```
function ForgotPasswordPage() {
  const [email, setEmail] = useState('');
```

```
  const [submitted, setSubmitted] = useState(false);

  const handleSubmit = (e: React.FormEvent) => {
    e.preventDefault();
    // Simulăm trimiterea
    setSubmitted(true);
  };

  if (submitted) {
    return (
      <div className="text-center p-8">
        <h2>Email trimis! ✉️</h2>
        <p>Verifică inbox-ul pentru linkul de resetare.</p>
        <Link to="/login">Înapoi la Login</Link>
      </div>
    );
  }

  return (
    <form onSubmit={handleSubmit}>
      <h1>Ai uitat parola?</h1>
      <input
        type="email"
        value={email}
        onChange={(e) => setEmail(e.target.value)}
        placeholder="Email"
        required
      />
      <button type="submit">Trimite link de resetare</button>
    </form>
  );
}
```

Task 1.4: Protected Routes și Layouts

Timp estimat: 8-10 ore | **Dificultate:** 🟡 Mediu

Ce vei construi: Un sistem care verifică dacă userul e logat și îl redirecționează dacă nu.

Cerințe:

- [ ] Component `ProtectedRoute` care verifică autentificarea
- [ ] Redirect la /login dacă userul nu e autentificat
- [ ] 3 layout-uri diferite: ClientLayout, OperatorLayout, AdminLayout
- [ ] Fiecare layout are sidebar-ul său specific

Cum funcționează (explicație):

```
User încearcă să acceseze /dashboard
      ↓
ProtectedRoute verifică: e logat?
      ↓
   NU → Redirect la /login
   DA → Verifică rolul
      ↓
   Client → ClientLayout + Dashboard
   Operator → OperatorLayout + Dashboard
   Admin → AdminLayout + Dashboard
```

Cod pentru ProtectedRoute:

```
interface ProtectedRouteProps {
  children: React.ReactNode;
  allowedRoles?: ('client' | 'operator' | 'admin')[];
}

function ProtectedRoute({ children, allowedRoles }: ProtectedRouteProps) {
  // Citim user-ul din localStorage (simplificat)
  const user = JSON.parse(localStorage.getItem('user') || 'null');

  // Nu e logat? -> Login
  if (!user) {
    return <Navigate to="/login" />;
  }

  // Are rol permis?
  if (allowedRoles && !allowedRoles.includes(user.role)) {
    return <Navigate to="/unauthorized" />;
  }

  return <>{children}</>;
}
```

Cum să simulezi autentificarea:

```
// La login success, salvează în localStorage:
localStorage.setItem('user', JSON.stringify({
  id: 1,
  name: 'Ion Popescu',
  email: 'ion@test.com',
  role: 'client' // sau 'operator' sau 'admin'
}));
```

```
// La logout:
localStorage.removeItem('user');
```

📊 2. SCORING & DECISION ENGINE

Task 2.1: Chestionar Scoring (Simplificat)

Timp estimat: 10-12 ore | **Dificultate:** 🟡 Mediu

Ce vei construi: Un formular în pași pentru colectarea informațiilor financiare.

Cerințe (reduse la 4 steps):

- [ ] Step 1: Venit lunar (salariu net, alte venituri)
- [ ] Step 2: Cheltuieli (chirie, utilități, rate existente)
- [ ] Step 3: Situație locuință (proprietar/chiriaș)
- [ ] Step 4: Rezumat
- [ ] Progress bar vizual
- [ ] Salvare automată în localStorage

Hint pentru auto-save:

```
// Salvează la fiecare modificare
useEffect(() => {
  localStorage.setItem('scoring-draft', JSON.stringify(formData));
}, [formData]);

// Încarcă la mount
useEffect(() => {
  const saved = localStorage.getItem('scoring-draft');
  if (saved) {
    setFormData(JSON.parse(saved));
  }
}, []);
```

Task 2.2: Calculator Scoring (Simplificat)

Timp estimat: 8-10 ore | **Dificultate:** 🟡 Mediu

Ce vei construi: Un tool interactiv care calculează și afișează scorul în timp real.

Cerințe:

- [ ] Input-uri: salariu, cheltuieli lunare, datorii existente
- [ ] Scor calculat automat (0-100)
- [ ] Bară colorată: roșu (0-40), galben (41-70), verde (71-100)
- [ ] Explicație text ce înseamnă scorul

Formula simplă de scoring:

```
function calculateScore(salariu: number, cheltuieli: number, datorii: number): number {
  // Calculăm rata de îndatorare
  const rataIndatorare = (cheltuieli + datorii) / salariu;

  // Scor bazat pe rata de îndatorare
  // < 30% = scor mare, > 60% = scor mic
  if (rataIndatorare < 0.3) return 85;
  if (rataIndatorare < 0.4) return 70;
  if (rataIndatorare < 0.5) return 55;
  if (rataIndatorare < 0.6) return 40;
  return 25;
}
```

Componentă pentru bara colorată:

```
function ScoreBar({ score }: { score: number }) {
  const getColor = () => {
    if (score >= 71) return 'bg-green-500';
    if (score >= 41) return 'bg-yellow-500';
    return 'bg-red-500';
  };

  return (
    <div className="w-full h-4 bg-gray-200 rounded">
      <div
        className={`h-full rounded ${getColor()}`}
        style={{ width: `${score}%` }}
      />
    </div>
  );
}
```

Task 2.3: Card Rezultat Decizie

Timp estimat: 4-6 ore | **Dificultate:** 🟢 Ușor

Ce vei construi: O componentă care afișează rezultatul: Aprobat, Respins, sau În așteptare.

Cerințe:

- [ ] Card mare cu rezultatul (icon + text)
- [ ] Culori diferite: verde (aprobat), roșu (respins), galben (pending)
- [ ] Scorul obținut
- [ ] Mesaj explicativ
- [ ] Buton pentru next step

Cod complet:

```
type DecisionStatus = 'approved' | 'rejected' | 'pending';

interface DecisionCardProps {
  status: DecisionStatus;
  score: number;
}

function DecisionCard({ status, score }: DecisionCardProps) {
  const config = {
    approved: {
      icon: '✅',
      title: 'Felicitări! Ești aprobat!',
      color: 'bg-green-100 border-green-500',
      message: 'Poți continua cu aplicația de credit.',
      buttonText: 'Aplică pentru credit',
    },
    rejected: {
      icon: '❌',
      title: 'Ne pare rău',
      color: 'bg-red-100 border-red-500',
      message: 'Scorul tău nu îndeplinește criteriile minime.',
      buttonText: 'Vezi sugestii de îmbunătățire',
    },
    pending: {
      icon: '⏳',
      title: 'În verificare',
      color: 'bg-yellow-100 border-yellow-500',
      message: 'Aplicația ta necesită verificare manuală.',
      buttonText: 'Contactează suport',
    },
  };

  const { icon, title, color, message, buttonText } = config[status];

  return (
    <div className={`p-6 rounded-lg border-2 ${color}`}>
      <div className="text-4xl mb-4">{icon}</div>
      <h2 className="text-xl font-bold">{title}</h2>
```

```
      <p className="text-gray-600 my-4">Scorul tău: {score}/100</p>
      <p>{message}</p>
      <button className="mt-4 px-4 py-2 bg-blue-500 text-white rounded">
        {buttonText}
      </button>
    </div>
  );
}
```

Task 2.4: Tracker Status KYC/AML

Timp estimat: 6-8 ore | **Dificultate:** 🟡 Mediu

Ce vei construi: Un timeline vizual care arată progresul verificărilor.

Cerințe:

- [ ] Timeline vertical cu 4 pași: Trimis → În verificare → Verificat/Respins
- [ ] Fiecare pas are icon, titlu, și dată
- [ ] Pasul curent e evidențiat
- [ ] Badge-uri colorate pentru status

Componentă Timeline:

```
interface TimelineStep {
  id: number;
  title: string;
  status: 'completed' | 'current' | 'pending';
  date?: string;
}

function VerificationTimeline({ steps }: { steps: TimelineStep[] }) {
  return (
    <div className="space-y-4">
      {steps.map((step, index) => (
        <div key={step.id} className="flex gap-4">
          {/* Linia și cercul */}
          <div className="flex flex-col items-center">
            <div className={`w-8 h-8 rounded-full flex items-center justify-center
              ${step.status === 'completed' ? 'bg-green-500 text-white' : ''}
              ${step.status === 'current' ? 'bg-blue-500 text-white' : ''}
              ${step.status === 'pending' ? 'bg-gray-300' : ''}
            `}>
              {step.status === 'completed' ? '✓' : index + 1}
            </div>
            {index < steps.length - 1 && (
              <div className="w-0.5 h-8 bg-gray-300" />
```

```
          )}
        </div>

        {/* Conținut */}
        <div>
          <h3 className="font-medium">{step.title}</h3>
          {step.date && <p className="text-sm text-gray-500">{step.date}</p>}
        </div>
      </div>
    ))}
  </div>
 );
}
```

👤 3. CLIENT PORTAL

Task 3.1: Dashboard Client

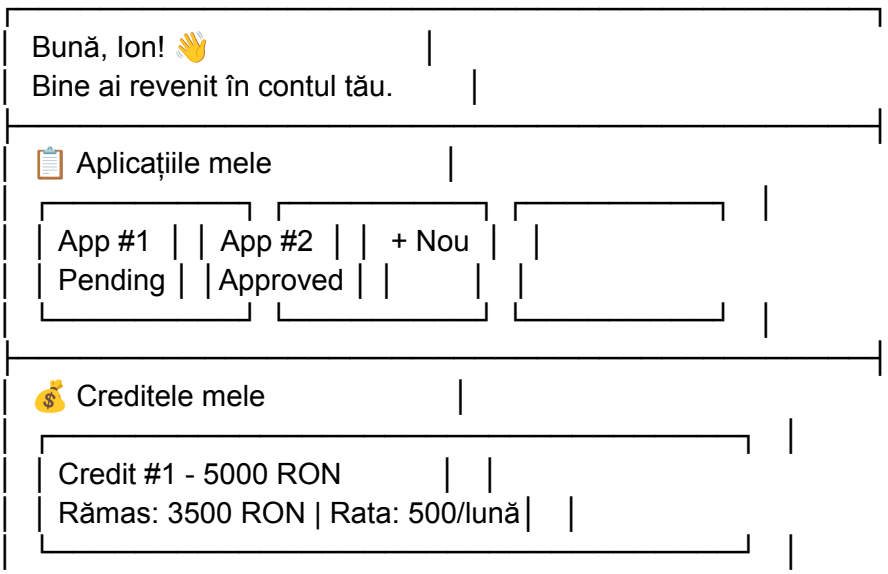
Timp estimat: 10-12 ore | **Dificultate:** 🟡 Mediu

Ce vei construi: Pagina principală pe care o vede clientul după login.

Cerințe (simplificate):

- [ ] Card cu salut și numele userului
- [ ] 2-3 carduri cu aplicațiile active (sau mesaj "Nu ai aplicații")
- [ ] 2-3 carduri cu creditele active (sau mesaj "Nu ai credite")
- [ ] 3 butoane rapide: "Aplică pentru credit", "Documentele mele", "Ajutor"

Structura paginii:



```
┌─────────────────────────────────────────────────────────┐
│  [Aplică] [Documente] [Ajutor]                          │
└─────────────────────────────────────────────────────────┘
```

Mock data:

```
const mockApplications = [
  { id: 1, amount: 5000, status: 'pending', date: '2025-01-10' },
  { id: 2, amount: 10000, status: 'approved', date: '2025-01-05' },
];

const mockCredits = [
  { id: 1, totalAmount: 5000, remainingAmount: 3500, monthlyPayment: 500 },
];
```

Task 3.2: Formular Aplicație Credit (Simplificat)

Timp estimat: 10-12 ore | **Dificultate:** 🟡 Mediu

Ce vei construi: Formularul prin care clientul aplică pentru un credit nou.

Cerințe (reduse la 4 steps):

- [ ] Step 1: Suma dorită (slider 1000-50000) și perioada (3-84 luni)
- [ ] Step 2: Informații venit (salariu, angajator)
- [ ] Step 3: Upload document (doar UI, nu funcțional complet)
- [ ] Step 4: Rezumat și submit
- [ ] Calculator vizibil: arată rata lunară estimată

Formula rată lunară:

```
function calculateMonthlyPayment(
  amount: number,
  months: number,
  interestRate: number = 0.15 // 15% anual
): number {
  const monthlyRate = interestRate / 12;
  const payment = amount * (monthlyRate * Math.pow(1 + monthlyRate, months))
                / (Math.pow(1 + monthlyRate, months) - 1);
  return Math.round(payment);
}

// Exemplu: 10000 RON pe 12 luni = ~902 RON/lună
```

Task 3.3: Pagina Documente (Simplificat)

Timp estimat: 8-10 ore | **Dificultate:** 🟡 Mediu

Ce vei construi: O pagină unde clientul își vede și încarcă documentele.

Cerințe:

- [ ] Lista documentelor existente (mock)
- [ ] Buton "Încarcă document" care deschide file picker
- [ ] Validare: doar PDF, JPG, PNG, max 10MB
- [ ] Afișare nume fișier după selectare
- [ ] Buton șterge document

Componentă Upload simplă:

```
function DocumentUploader() {
  const [files, setFiles] = useState<File[]>([]);
  const [error, setError] = useState('');

  const handleFileChange = (e: React.ChangeEvent<HTMLInputElement>) => {
    const file = e.target.files?.[0];
    if (!file) return;

    // Validare tip
    const allowedTypes = ['application/pdf', 'image/jpeg', 'image/png'];
    if (!allowedTypes.includes(file.type)) {
      setError('Doar PDF, JPG sau PNG sunt permise');
      return;
    }

    // Validare dimensiune (10MB)
    if (file.size > 10 * 1024 * 1024) {
      setError('Fișierul e prea mare (max 10MB)');
      return;
    }

    setError('');
    setFiles([...files, file]);
  };

  return (
    <div>
      <input
        type="file"
        onChange={handleFileChange}
        accept=".pdf,.jpg,.jpeg,.png"
      />
```

```
      {error && <p className="text-red-500">{error}</p>}

      <ul>
        {files.map((file, i) => (
          <li key={i}>
            {file.name} ({(file.size / 1024).toFixed(0)} KB)
            <button onClick={() => setFiles(files.filter((_, j) => j !== i))}>
              🗑️
            </button>
          </li>
        ))}
      </ul>
    </div>
  );
}
```

Task 3.4: Notificări (Simplificat)

Timp estimat: 6-8 ore | **Dificultate:** 🟢 Ușor-Mediu

Ce vei construi: Un sistem simplu de notificări cu dropdown.

Cerințe:

- [ ] Icon clopoțel în header cu număr de notificări necitite
- [ ] Click pe icon → dropdown cu lista notificărilor
- [ ] Fiecare notificare: icon, text, timestamp
- [ ] Buton "Marchează toate ca citite"

Componentă NotificationBell:

```
const mockNotifications = [
  { id: 1, text: 'Aplicația ta a fost aprobată!', read: false, date: '2025-01-10' },
  { id: 2, text: 'Document nou solicitat', read: false, date: '2025-01-09' },
  { id: 3, text: 'Plată recepționată', read: true, date: '2025-01-08' },
];

function NotificationBell() {
  const [isOpen, setIsOpen] = useState(false);
  const [notifications, setNotifications] = useState(mockNotifications);

  const unreadCount = notifications.filter(n => !n.read).length;

  const markAllRead = () => {
    setNotifications(notifications.map(n => ({ ...n, read: true })));
  };
```

```
  return (
    <div className="relative">
      <button onClick={() => setIsOpen(!isOpen)}>
        🔔
        {unreadCount > 0 && (
          <span className="absolute -top-1 -right-1 bg-red-500 text-white
                           text-xs rounded-full w-5 h-5">
            {unreadCount}
          </span>
        )}
      </button>

      {isOpen && (
        <div className="absolute right-0 mt-2 w-80 bg-white shadow-lg rounded-lg">
          <div className="p-2 border-b flex justify-between">
            <span>Notificări</span>
            <button onClick={markAllRead} className="text-blue-500 text-sm">
              Marchează citite
            </button>
          </div>
          {notifications.map(n => (
            <div key={n.id} className={`p-3 border-b ${!n.read ? 'bg-blue-50' : ''}`}>
              <p>{n.text}</p>
              <span className="text-xs text-gray-500">{n.date}</span>
            </div>
          ))}
        </div>
      )}
    </div>
  );
}
```

👨‍💼 4. OPERATOR DASHBOARD

Task 4.1: Tabel Aplicații

Timp estimat: 10-12 ore | **Dificultate:** 🟡 Mediu

Ce vei construi: Un tabel cu toate aplicațiile pentru operatori.

Cerințe (simplificate):

- [ ] Tabel cu coloane: Nume Client, Sumă, Data, Status, Acțiuni
- [ ] Filtrare după status (dropdown)
- [ ] Butoane acțiuni: Aprobă, Respinge, Vezi detalii

- [ ] Click pe rând → modal cu detalii complete

Mock data și tabel:

```
const mockApplications = [
  { id: 1, customerName: 'Ion Popescu', amount: 5000, date: '2025-01-10', status: 'pending' },
  { id: 2, customerName: 'Maria Ionescu', amount: 15000, date: '2025-01-09', status:
'pending' },
  { id: 3, customerName: 'Andrei Popa', amount: 8000, date: '2025-01-08', status: 'approved'
},
];

function ApplicationsTable() {
  const [filter, setFilter] = useState('all');
  const [applications, setApplications] = useState(mockApplications);

  const filtered = filter === 'all'
    ? applications
    : applications.filter(a => a.status === filter);

  const handleApprove = (id: number) => {
    setApplications(apps =>
      apps.map(a => a.id === id ? { ...a, status: 'approved' } : a)
    );
  };

  return (
    <div>
      <select value={filter} onChange={e => setFilter(e.target.value)}>
        <option value="all">Toate</option>
        <option value="pending">În așteptare</option>
        <option value="approved">Aprobate</option>
        <option value="rejected">Respinse</option>
      </select>

      <table className="w-full mt-4">
        <thead>
          <tr>
            <th>Client</th>
            <th>Sumă</th>
            <th>Data</th>
            <th>Status</th>
            <th>Acțiuni</th>
          </tr>
        </thead>
        <tbody>
          {filtered.map(app => (
            <tr key={app.id}>
```

```
            <td>{app.customerName}</td>
            <td>{app.amount} RON</td>
            <td>{app.date}</td>
            <td>{app.status}</td>
            <td>
              <button onClick={() => handleApprove(app.id)}>✓ </button>
              <button> ✗ </button>
              <button>👁️</button>
            </td>
          </tr>
        ))}
      </tbody>
    </table>
  </div>
 );
}
```

Task 4.2: Dashboard Monitorizare (Simplificat)

Timp estimat: 8-10 ore | **Dificultate:** 🟡 Mediu

Ce vei construi: Dashboard cu statistici și grafice pentru operatori.

Cerințe:

- [ ] 4 carduri KPI: Total aplicații, Aprobate, Respinse, În așteptare
- [ ] 1 grafic pie/doughnut cu distribuția statusurilor
- [ ] Design responsive (carduri în grid)

Exemplu KPI Card:

```
interface KpiCardProps {
  title: string;
  value: number;
  icon: string;
  color: string;
}

function KpiCard({ title, value, icon, color }: KpiCardProps) {
  return (
    <div className={`p-4 rounded-lg ${color}`}>
      <div className="text-2xl">{icon}</div>
      <div className="text-3xl font-bold">{value}</div>
      <div className="text-sm">{title}</div>
    </div>
  );
}
```

```
// Utilizare:
<div className="grid grid-cols-4 gap-4">
  <KpiCard title="Total" value={150} icon="📊" color="bg-blue-100" />
  <KpiCard title="Aprobate" value={80} icon="✅" color="bg-green-100" />
  <KpiCard title="Respinse" value={30} icon="❌" color="bg-red-100" />
  <KpiCard title="În așteptare" value={40} icon="⏳" color="bg-yellow-100" />
</div>
```

Pentru grafic - folosește Recharts:

```
import { PieChart, Pie, Cell } from 'recharts';

const data = [
  { name: 'Aprobate', value: 80 },
  { name: 'Respinse', value: 30 },
  { name: 'În așteptare', value: 40 },
];
const COLORS = ['#22c55e', '#ef4444', '#eab308'];

<PieChart width={300} height={300}>
  <Pie data={data} dataKey="value" cx="50%" cy="50%" outerRadius={100}>
    {data.map((_, index) => (
      <Cell key={index} fill={COLORS[index]} />
    ))}
  </Pie>
</PieChart>
```

Task 4.3: Pagina Colectări (Simplificat)

Timp estimat: 8-10 ore | **Dificultate:** 🟡 Mediu

Ce vei construi: Tabel cu creditele restante pentru urmărire plăți.

Cerințe:

- [ ] Tabel: Client, Sumă credit, Rămas, Zile întârziere, Status
- [ ] Color coding pe rânduri: verde (la zi), galben (1-30 zile), roșu (30+ zile)
- [ ] Filtru după zile întârziere
- [ ] Buton "Trimite reminder" (doar UI)

Logica pentru culori:

```
function getRowColor(daysOverdue: number): string {
  if (daysOverdue === 0) return 'bg-green-50';
  if (daysOverdue <= 30) return 'bg-yellow-50';
```

```
  return 'bg-red-50';
}
```

Task 4.4: Management Clienți (Simplificat)

Timp estimat: 8-10 ore | **Dificultate:** 🟡 Mediu

Ce vei construi: Director cu toți clienții și detaliile lor.

Cerințe:

- [ ] Tabel clienți: Nume, Email, Telefon, Status KYC, Acțiuni
- [ ] Search după nume sau email
- [ ] Click pe client → card cu detalii
- [ ] Badge-uri pentru status KYC (Verified/Pending/Rejected)

⚙️ 5. ADMIN CONSOLE

Task 5.1: Management Utilizatori

Timp estimat: 10-12 ore | **Dificultate:** 🟡 Mediu

Ce vei construi: Panel pentru gestionarea utilizatorilor platformei.

Cerințe (simplificate):

- [ ] Tabel utilizatori: Nume, Email, Rol, Status, Acțiuni
- [ ] Buton "Adaugă utilizator" → modal cu formular
- [ ] Dropdown pentru schimbare rol
- [ ] Toggle active/inactive
- [ ] Filtru după rol

Formular adăugare user:

```
interface UserFormData {
  name: string;
  email: string;
  role: 'client' | 'operator' | 'admin';
}

function AddUserModal({ onClose, onSave }) {
  const [formData, setFormData] = useState<UserFormData>({
    name: '',
    email: '',
```

```
    role: 'client',
  });

  return (
    <div classNam="fixed inset-0 bg-black/50 flex items-center justify-center">
      <div classNam="bg-white p-6 rounded-lg w-96">
        <h2>Adaugă utilizator</h2>

        <input
          placeholder="Nume"
          value={formData.name}
          onChange={e => setFormData({...formData, name: e.target.value})}
        />

        <input
          placeholder="Email"
          type="email"
          value={formData.email}
          onChange={e => setFormData({...formData, email: e.target.value})}
        />

        <select
          value={formData.role}
          onChange={e => setFormData({...formData, role: e.target.value})}
        >
          <option value="client">Client</option>
          <option value="operator">Operator</option>
          <option value="admin">Admin</option>
        </select>

        <div classNam="flex gap-2 mt-4">
          <button onClick={onClose}>Anulează</button>
          <button onClick={() => onSave(formData)}>Salvează</button>
        </div>
      </div>
    </div>
  );
}
```

Task 5.2: Setări Sistem (Simplificat)

Timp estimat: 6-8 ore | **Dificultate:** 🟢 Ușor-Mediu

Ce vei construi: Pagină de configurare cu setări grupate în tabs.

Cerințe:

- [ ] 3 tabs: General, Creditare, Notificări
- [ ] General: Nume companie, timezone (dropdown)
- [ ] Creditare: Sumă min/max (inputs), Dobândă (slider)
- [ ] Notificări: Toggle-uri pentru email/SMS
- [ ] Buton Save cu toast success

Componenta Tabs:

```
function SettingsTabs() {
  const [activeTab, setActiveTab] = useState('general');

  return (
    <div>
      <div className="flex gap-2 border-b">
        {['general', 'creditare', 'notificari'].map(tab => (
          <button
            key={tab}
            onClick={() => setActiveTab(tab)}
            className={activeTab === tab ? 'border-b-2 border-blue-500' : ''}
          >
            {tab.charAt(0).toUpperCase() + tab.slice(1)}
          </button>
        ))}
      </div>

      <div className="p-4">
        {activeTab === 'general' && <GeneralSettings />}
        {activeTab === 'creditare' && <LendingSettings />}
        {activeTab === 'notificari' && <NotificationSettings />}
      </div>
    </div>
  );
}
```

Task 5.3: Audit Log (Simplificat)

Timp estimat: 6-8 ore | **Dificultate:** 🟢 Ușor-Mediu

Ce vei construi: Tabel cu istoricul acțiunilor din platformă.

Cerințe:

- [ ] Tabel: Data/Ora, Utilizator, Acțiune, Detalii
- [ ] Filtru după tip acțiune (Login, Create, Update, Delete)
- [ ] Filtru după dată (date picker)
- [ ] Sorting desc după timestamp

Mock audit logs:

```
const mockAuditLogs = [
  { id: 1, timestamp: '2025-01-10 14:30', user: 'Admin', action: 'CREATE', details: 'Created user Ion' },
  { id: 2, timestamp: '2025-01-10 14:25', user: 'Operator1', action: 'UPDATE', details: 'Approved application #123' },
  { id: 3, timestamp: '2025-01-10 14:20', user: 'Admin', action: 'DELETE', details: 'Deleted document #456' },
];
```

Task 5.4: Rapoarte (Simplificat)

Timp estimat: 8-10 ore | **Dificultate:** 🟡 Mediu

Ce vei construi: Generator de rapoarte cu grafice.

Cerințe:

- [ ] Dropdown pentru tip raport (Aplicații, Colectări, Operațiuni)
- [ ] Date picker pentru perioadă
- [ ] Buton "Generează"
- [ ] Afișare grafic linie/bar după generare
- [ ] Buton "Export CSV" (simulat - doar alert)

📚 Resurse Generale

Cum să începi orice task:

1. **Citește cerințele** complet înainte să scrii cod
2. **Desenează pe hârtie** cum ar trebui să arate
3. **Creează fișierele** în locația corectă
4. **Începe simplu** - fă să funcționeze, apoi îmbunătățește
5. **Testează des** - după fiecare feature mică

Succes! 🚀