```
import React, { useState, useEffect, useMemo, useRef } from 'react';

// --- Firebase Imports ---
// This app uses Firebase to simulate a backend for data persistence.
import { initializeApp } from "firebase/app";
import {
getAuth,
signInAnonymously,
onAuthStateChanged,
signInWithCustomToken
} from "firebase/auth";
import {
getFirestore,
doc,
setDoc,
getDoc,
onSnapshot,
collection,
addDoc,
updateDoc,
increment,
writeBatch,
Timestamp,
query,
where,
getDocs,
setLogLevel
} from "firebase/firestore";

// --- Icon Imports ---
// Using lucide-react for a clean icon set
import {
LayoutDashboard,
TrendingUp,
Star,
Wallet,
CheckCircle,
XCircle,
Clock,
ShieldCheck,
User,
ArrowRight,
```

```
Loader2,
Database,
Banknote,
Activity
} from 'lucide-react';

// --- Firebase Configuration ---
// These global variables are expected to be injected by the environment.
const firebaseConfig = typeof firebase_config !== 'undefined'
? JSON.parse(firebase_config)
: { apiKey: "YOUR_FALLBACK_API_KEY", authDomain: "...", projectId: "..." };

const appId = typeof app_id !== 'undefined' ? app_id : 'coingrow-default';

// --- App Initialization ---
let app;
let auth;
let db;

try {
app = initializeApp(firebaseConfig);
auth = getAuth(app);
db = getFirestore(app);
// Enable debug logging for Firestore
setLogLevel('debug');
} catch (e) {
console.error("Firebase initialization error:", e);
}

// --- App Constants ---

const CORE_PLANS = [
{ id: 'starter', name: 'Starter', investment: 399, dailyEarning: 13, icon: Activity },
{ id: 'pro', name: 'Pro', investment: 899, dailyEarning: 29, icon: TrendingUp },
{ id: 'expert', name: 'Expert', investment: 2999, dailyEarning: 99, icon: Banknote },
{ id: 'elite', name: 'Elite', investment: 15000, dailyEarning: 400, icon: Database },
];

const VIP_PLANS = [
{ id: 'vip1', level: 1, name: 'Quick Flip', investment: 200, termDays: 9, totalReturn: 400,
depositRequired: 499 },
{ id: 'vip2', level: 2, name: 'Rapid Growth', investment: 999, termDays: 9, totalReturn: 1600,
```

```
depositRequired: 1000 },
{ id: 'vip3', level: 3, name: 'Power Surge', investment: 1900, termDays: 9, totalReturn: 3500,
depositRequired: 4500 },
{ id: 'vip4', level: 4, name: 'Mega Boost', investment: 8000, termDays: 2, totalReturn: 16000,
depositRequired: 16000 },
];

const MIN_DEPOSIT = 300;
const MIN_WITHDRAWAL = 425;
const MAX_WITHDRAWAL = 50000;

// --- Helper Functions ---

/**
* Formats a number as Indian Rupees (INR).
* @param {number} amount - The number to format.
* @returns {string} - Formatted currency string.
*/
const formatCurrency = (amount) => {
return new Intl.NumberFormat('en-IN', {
style: 'currency',
currency: 'INR',
minimumFractionDigits: 2,
maximumFractionDigits: 2,
}).format(amount || 0);
};

/**
* Calculates VIP level and progress based on total deposits.
* @param {number} totalDeposits - The user's total deposited amount.
* @returns {object} - { level, progress, nextLevelAmount }
*/
const getVipInfo = (totalDeposits) => {
let level = 0;
let progress = 0;
let nextLevelAmount = VIP_PLANS[0].depositRequired;

if (totalDeposits >= VIP_PLANS[3].depositRequired) {
level = 4;
progress = 100;
nextLevelAmount = VIP_PLANS[3].depositRequired;
} else if (totalDeposits >= VIP_PLANS[2].depositRequired) {
```

```
level = 3;
const prevLevelAmount = VIP_PLANS[2].depositRequired;
nextLevelAmount = VIP_PLANS[3].depositRequired;
progress = ((totalDeposits - prevLevelAmount) / (nextLevelAmount - prevLevelAmount)) * 100;
} else if (totalDeposits >= VIP_PLANS[1].depositRequired) {
level = 2;
const prevLevelAmount = VIP_PLANS[1].depositRequired;
nextLevelAmount = VIP_PLANS[2].depositRequired;
progress = ((totalDeposits - prevLevelAmount) / (nextLevelAmount - prevLevelAmount)) * 100;
} else if (totalDeposits >= VIP_PLANS[0].depositRequired) {
level = 1;
const prevLevelAmount = 0; // Starting from 0
nextLevelAmount = VIP_PLANS[1].depositRequired;
progress = ((totalDeposits - prevLevelAmount) / (nextLevelAmount - prevLevelAmount)) * 100;
} else {
level = 0;
nextLevelAmount = VIP_PLANS[0].depositRequired;
progress = (totalDeposits / nextLevelAmount) * 100;
}

return { level, progress: Math.min(100, Math.max(0, progress)), nextLevelAmount,
currentLevelAmount: VIP_PLANS[level-1]?.depositRequired || 0 };
};

// --- React Components ---

/**
* Header Component
* Displays app name, user's VIP level, and balance.
*/
const Header = ({ userProfile, vipInfo }) => {
return (
<header className="sticky top-0 z-20 flex items-center justify-between p-4 bg-gray-900
shadow-lg border-b border-gray-700">
<div>
<h1 className="text-2xl font-bold text-emerald-400">CoinGrow</h1>
<p className="text-sm text-gray-400">Start Small, Grow Steadily.</p>
</div>
<div className="text-right">
<div className="flex items-center justify-end gap-1.5 text-yellow-400">
<Star className="w-5 h-5" />
<span className="font-bold text-lg">VIP {vipInfo.level}</span>
```

```
</div>
<p className="text-lg font-semibold text-white">
{formatCurrency(userProfile.availableBalance)}
</p>
</div>
</header>
);
};

/**
* Bottom Navigation Component
* Allows user to switch between main views.
*/
const BottomNav = ({ currentView, setView }) => {
const navItems = [
{ id: 'dashboard', label: 'Dashboard', icon: LayoutDashboard },
{ id: 'invest', label: 'Invest', icon: TrendingUp },
{ id: 'vip', label: 'VIP', icon: Star },
{ id: 'wallet', label: 'Wallet', icon: Wallet },
];

return (
<nav className="sticky bottom-0 z-20 grid grid-cols-4 gap-2 p-3 bg-gray-900 shadow-inner
border-t border-gray-700">
{navItems.map((item) => (
<button
key={item.id}
onClick={() => setView(item.id)}
className={`flex flex-col items-center justify-center p-2 rounded-lg transition-colors ${
currentView === item.id
? 'bg-emerald-600 text-white'
: 'text-gray-400 hover:bg-gray-800'
}`}
>
<item.icon className="w-6 h-6 mb-1" />
<span className="text-xs font-medium">{item.label}</span>
</button>
))}
</nav>
);
};
```

```
/**
* Dashboard View
* Main screen showing balance summary and active plans.
*/
const Dashboard = ({ userProfile, activePlans, activeVipPlans, vipInfo }) => {
const totalDailyEarnings = useMemo(() => {
return activePlans.reduce((sum, plan) => sum + plan.dailyEarning, 0);
}, [activePlans]);

const activeInvestmentAmount = useMemo(() => {
const core = activePlans.reduce((sum, plan) => sum + plan.investment, 0);
const vip = activeVipPlans
.filter(p => !p.isMatured)
.reduce((sum, plan) => sum + plan.investment, 0);
return core + vip;
}, [activePlans, activeVipPlans]);

return (
<div className="space-y-6 p-4">
{/ Simulation Note /}
<div className="bg-yellow-800 border border-yellow-600 text-yellow-100 p-3 rounded-lg
text-center">
<div className="flex items-center justify-center gap-2">
<Clock className="w-5 h-5" />
<span className="font-medium">Simulation Active</span>
</div>
<p className="text-sm">Earnings are calculated every 10 seconds to simulate a 24-hour
cycle.</p>
</div>

{/ Balance Cards /}
<div className="grid grid-cols-2 gap-4">
<InfoCard title="Available Balance" value={formatCurrency(userProfile.availableBalance)}
icon={Wallet} color="text-white" />
<InfoCard title="Total Earnings" value={formatCurrency(userProfile.totalEarnings)}
icon={TrendingUp}color="text-emerald-400" />
<InfoCard title="Active Investments" value={formatCurrency(activeInvestmentAmount)}
icon={Database} color="text-blue-400" />
<InfoCard title="Total Daily Income" value={formatCurrency(totalDailyEarnings)} icon={Activity}
color="text-emerald-400" />
</div>
```

```
{/ VIP Progress /}
<div className="bg-gray-800 p-4 rounded-lg shadow-lg">
<div className="flex justify-between items-center mb-2">
<h3 className="text-lg font-semibold text-yellow-400">VIP {vipInfo.level}</h3>
<span className="text-sm text-gray-400">Next:
{formatCurrency(vipInfo.nextLevelAmount)}</span>
</div>
<div className="w-full bg-gray-700 rounded-full h-2.5">
<div
className="bg-yellow-400 h-2.5 rounded-full transition-all duration-500"
style={{ width: `${vipInfo.progress}%` }}
></div>
</div>
<p className="text-xs text-gray-400 mt-2 text-center">
Deposit {formatCurrency(vipInfo.nextLevelAmount - userProfile.totalDeposits)} more to reach
VIP {vipInfo.level + 1}.
</p>
</div>

{/ Active Core Plans /}
<div className="space-y-3">
<h3 className="text-xl font-semibold text-white">Active Daily Plans</h3>
{activePlans.length === 0 ? (
<p className="text-gray-400 text-center p-4 bg-gray-800 rounded-lg">No active daily plans.
Visit the 'Invest' tab to start.</p>
) : (
activePlans.map((plan, index) => (
<PlanCard key={`core-${index}`} plan={plan} isVip={false} />
))
)}
</div>

{/ Active VIP Plans /}
<div className="space-y-3">
<h3 className="text-xl font-semibold text-white">Active VIP Plans</h3>
{activeVipPlans.filter(p => !p.isMatured).length === 0 ? (
<p className="text-gray-400 text-center p-4 bg-gray-800 rounded-lg">No active VIP plans.
Visit the 'VIP' tab to unlock.</p>
) : (
activeVipPlans.filter(p => !p.isMatured).map((plan) => (
<PlanCard key={plan.id} plan={plan} isVip={true} />
))
```

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

/**
* Info Card Component (for Dashboard)
*/
const InfoCard = ({ title, value, icon: Icon, color }) => (
<div className="bg-gray-800 p-4 rounded-lg shadow-lg flex items-center gap-4">
<div className={`p-2 bg-gray-700 rounded-full ${color}`}>
<Icon className="w-6 h-6" />
</div>
<div>
<p className="text-sm text-gray-400">{title}</p>
<p className={`text-xl font-bold ${color}`}>{value}</p>
</div>
</div>
);

/**
* Plan Card Component (for Dashboard)
*/
const PlanCard = ({ plan, isVip }) => {
const [timeLeft, setTimeLeft] = useState("");

useEffect(() => {
if (isVip && plan.endDate) {
const timer = setInterval(() => {
const now = new Date().getTime();
const end = plan.endDate.toMillis();
const distance = end - now;

if (distance < 0) {
setTimeLeft("Matured");
clearInterval(timer);
return;
}

const days = Math.floor(distance / (1000  60  60 * 24));
const hours = Math.floor((distance % (1000  60  60  24)) / (1000  60 * 60));
```

```
const minutes = Math.floor((distance % (1000  60  60)) / (1000 * 60));
setTimeLeft(`${days}d ${hours}h ${minutes}m left`);
}, 1000);
return () => clearInterval(timer);
}
}, [plan, isVip]);

return (
<div className="bg-gray-800 p-4 rounded-lg shadow-lg">
<div className="flex justify-between items-center">
<div>
<h4 className="text-lg font-semibold text-white">{plan.planName}</h4>
<p className="text-sm text-gray-400">Invested: {formatCurrency(plan.investment)}</p>
</div>
<div className="text-right">
{isVip ? (
<>
<p className="text-lg font-bold text-emerald-400">{formatCurrency(plan.totalReturn)}</p>
<p className="text-sm text-yellow-400">{timeLeft}</p>
</>
) : (
<>
<p className="text-lg font-bold text-emerald-400">+{formatCurrency(plan.dailyEarning)}</p>
<p className="text-sm text-gray-400">/ day</p>
</>
)}
</div>
</div>
</div>
);
};


/**
* Invest View
* Displays available core investment plans.
*/
const Invest = ({ onInvest, userBalance }) => {
return (
<div className="p-4 space-y-4">
<h2 className="text-2xl font-semibold text-white mb-4">Core Investment Plans</h2>
<p className="text-gray-400 mb-6">Invest in a plan to start earning daily income. Your
```

```
principal is included in the plan and income is credited daily.</p>

<div className="space-y-4">
{CORE_PLANS.map((plan) => (
<div key={plan.id} className="bg-gray-800 p-4 rounded-lg shadow-lg border
border-gray-700">
<div className="flex items-center justify-between mb-3">
<div className="flex items-center gap-3">
<plan.icon className="w-8 h-8 text-emerald-400" />
<h3 className="text-xl font-bold text-white">{plan.name}</h3>
</div>
<span className="text-lg font-semibold
text-emerald-400">+{formatCurrency(plan.dailyEarning)}/ day</span>
</div>
<p className="text-gray-400 mb-4">Invest {formatCurrency(plan.investment)} and get
{formatCurrency(plan.dailyEarning)} credited to your wallet every day.</p>

<button
onClick={() => onInvest(plan)}
disabled={userBalance < plan.investment}
className="w-full bg-emerald-600 text-white font-bold py-3 px-4 rounded-lg transition-colors
hover:bg-emerald-700 disabled:bg-gray-600 disabled:cursor-not-allowed"
>
{userBalance < plan.investment ? `Insufficient Balance` : `Invest Now`}
</button>
</div>
))}
</div>
</div>
);
};

/**
* VIP View
* Displays VIP progress and exclusive short-term plans.
*/
const Vip = ({ onVipInvest, userBalance, vipInfo }) => {
return (
<div className="p-4 space-y-6">
<h2 className="text-2xl font-semibold text-white mb-4">VIP Center</h2>

{/ VIP Progress Card /}
```

```
<div className="bg-gradient-to-br from-yellow-500 to-yellow-700 p-5 rounded-lg shadow-lg text-gray-900">
<div className="flex justify-between items-center mb-2">
<h3 className="text-2xl font-bold">VIP {vipInfo.level}</h3>
<Star className="w-8 h-8" />
</div>
<p className="mb-3">Deposit {formatCurrency(vipInfo.nextLevelAmount - (vipInfo.currentLevelAmount || 0))} to reach the next level.</p>
<div className="w-full bg-yellow-200 rounded-full h-3">
<div
className="bg-gray-900 h-3 rounded-full"
style={{ width: `${vipInfo.progress}%` }}
></div>
</div>
<p className="text-xs text-right mt-1">
{formatCurrency(vipInfo.currentLevelAmount || 0)} /
{formatCurrency(vipInfo.nextLevelAmount)}
</p>
</div>

<h3 className="text-xl font-semibold text-white">Exclusive VIP Plans</h3>

<div className="space-y-4">
{VIP_PLANS.map((plan) => {
const isUnlocked = vipInfo.level >= plan.level;
return (
<div
key={plan.id}
className={`bg-gray-800 p-4 rounded-lg shadow-lg border ${
isUnlocked ? 'border-yellow-400' : 'border-gray-700 opacity-60'
}`}
>
<div className="flex items-center justify-between mb-3">
<h3 className="text-xl font-bold text-yellow-400">{plan.name}</h3>
<span className="px-3 py-1 text-xs font-bold bg-yellow-400 text-gray-900 rounded-full">
VIP {plan.level}
</span>
</div>
<p className="text-gray-300">Invest: <span
className="font-semibold">{formatCurrency(plan.investment)}</span></p>
<p className="text-gray-300">Term:<span className="font-semibold">{plan.termDays} Days</span></p>
```

```
<p className="text-gray-300 mb-4">Total Return: <span className="font-semibold
text-emerald-400">{formatCurrency(plan.totalReturn)}</span></p>

<button
onClick={() => onVipInvest(plan)}
disabled={!isUnlocked || userBalance < plan.investment}
className="w-full bg-yellow-500 text-gray-900 font-bold py-3 px-4 rounded-lg
transition-colors hover:bg-yellow-600 disabled:bg-gray-600 disabled:text-gray-400
disabled:cursor-not-allowed"
>
{!isUnlocked
? `Requires VIP ${plan.level}`
: userBalance < plan.investment
? 'Insufficient Balance'
: 'Invest Now'
}
</button>
</div>
);
})}
</div>
</div>
);
};

/**
* Wallet View
* Handles deposits, withdrawals, and KYC.
*/
const Wallet = ({ userProfile, onDeposit, onWithdraw, onKycSubmit, setShowKycModal }) => {
const [depositAmount, setDepositAmount] = useState(MIN_DEPOSIT);
const [withdrawAmount, setWithdrawAmount] = useState(MIN_WITHDRAWAL);

const kycStatusDisplay = {
'none': <span className="text-red-400">Not Verified</span>,
'pending': <span className="text-yellow-400">Pending Review</span>,
'verified': <span className="text-emerald-400">Verified</span>,
};

const handleWithdrawClick = () => {
if (userProfile.kycStatus !== 'verified') {
setShowKycModal(true);
```

```
} else {
onWithdraw(withdrawAmount);
setWithdrawAmount(MIN_WITHDRAWAL);
}
};

return (
<div className="p-4 space-y-6">
<h2 className="text-2xl font-semibold text-white mb-4">Wallet & KYC</h2>

{/ KYC Status /}
<div className="bg-gray-800 p-4 rounded-lg shadow-lg flex items-center justify-between">
<div className="flex items-center gap-3">
<ShieldCheck className="w-6 h-6 text-blue-400" />
<div>
<h3 className="text-lg font-semibold text-white">KYC Status</h3>
<p className="text-sm">{kycStatusDisplay[userProfile.kycStatus]}</p>
</div>
</div>
{userProfile.kycStatus === 'none' && (
<button
onClick={() => setShowKycModal(true)}
className="bg-blue-500 text-white font-medium py-2 px-4 rounded-lg text-sm"
>
Verify Now
</button>
)}
</div>

{/ Deposit /}
<div className="bg-gray-800 p-4 rounded-lg shadow-lg space-y-3">
<h3 className="text-xl font-semibold text-white">Deposit</h3>
<p className="text-sm text-gray-400">Minimum Deposit: {formatCurrency(MIN
_DEPOSIT)}</p>
<div>
<label htmlFor="deposit" className="block text-sm font-medium text-gray-300
mb-1">Amount (INR)</label>
<input
type="number"
id="deposit"
value={depositAmount}
onChange={(e) => setDepositAmount(Number(e.target.value))}
```

```
min={MIN_DEPOSIT}
className="w-full bg-gray-700 border border-gray-600 rounded-lg p-3 text-white text-lg"
/>
</div>
<button
onClick={() => {
onDeposit(depositAmount);
setDepositAmount(MIN_DEPOSIT);
}}
disabled={depositAmount < MIN_DEPOSIT}
className="w-full bg-emerald-600 text-white font-bold py-3 px-4 rounded-lg transition-colors
hover:bg-emerald-700 disabled:bg-gray-600"
>
Deposit {formatCurrency(depositAmount)}
</button>
</div>

{/ Withdraw /}
<div className="bg-gray-800 p-4 rounded-lg shadow-lg space-y-3">
<h3 className="text-xl font-semibold text-white">Withdraw</h3>
<p className="text-sm text-gray-400">Min: {formatCurrency(MINWITHDRAWAL)} | Max:
{formatCurrency(MAXWITHDRAWAL)}</p>
<div>
<label htmlFor="withdraw" className="block text-sm font-medium text-gray-300
mb-1">Amount (INR)</label>
<input
type="number"
id="withdraw"
value={withdrawAmount}
onChange={(e) => setWithdrawAmount(Number(e.target.value))}
min={MIN_WITHDRAWAL}
max={MAX_WITHDRAWAL}
className="w-full bg-gray-700 border border-gray-600 rounded-lg p-3 text-white text-lg"
/>
</div>
<button
onClick={handleWithdrawClick}
disabled={withdrawAmount < MINWITHDRAWAL || withdrawAmount >
userProfile.availableBalance || withdrawAmount > MAXWITHDRAWAL}
className="w-full bg-blue-500 text-white font-bold py-3 px-4 rounded-lg transition-colors
hover:bg-blue-600 disabled:bg-gray-600"
>
```

```
{userProfile.kycStatus !== 'verified'
? 'Verify KYC to Withdraw'
: withdrawAmount > userProfile.availableBalance
? 'Insufficient Balance'
: `Withdraw ${formatCurrency(withdrawAmount)}`
}
</button>
</div>
</div>
);
};

/**
* KYC Modal
* A modal for submitting KYC details.
*/
const KycModal = ({ onClose, onSubmit }) => {
const [pan, setPan] = useState('');
const [aadhaar, setAadhaar] = useState('');

const handleSubmit = (e) => {
e.preventDefault();
if (pan.length === 10 && aadhaar.length === 12) {
onSubmit(pan, aadhaar);
}
};

return (
<div className="fixed inset-0 z-50 bg-black bg-opacity-75 flex items-center justify-center
p-4">
<div className="bg-gray-800 w-full max-w-md p-6 rounded-lg shadow-xl">
<h2 className="text-2xl font-semibold text-white mb-4">KYC Verification</h2>
<p className="text-gray-400 mb-6">Verification is required for withdrawals. Please enter your
details.</p>
<form onSubmit={handleSubmit} className="space-y-4">
<div>
<label htmlFor="pan" className="block text-sm font-medium text-gray-300 mb-1">PAN
Number (10 digits)</label>
<input
type="text"
id="pan"
value={pan}
```

```
onChange={(e) => setPan(e.target.value.toUpperCase())}
maxLength={10}
className="w-full bg-gray-700 border border-gray-600 rounded-lg p-3 text-white"
/>
</div>
<div>
<label htmlFor="aadhaar" className="block text-sm font-medium text-gray-300
mb-1">Aadhaar Number (12 digits)</label>
<input
type="text"
id="aadhaar"
value={aadhaar}
onChange={(e) => setAadhaar(e.target.value.replace(/\D/g, ''))}
maxLength={12}
className="w-full bg-gray-700 border border-gray-600 rounded-lg p-3 text-white"
/>
</div>
<div className="flex gap-4 pt-4">
<button
type="button"
onClick={onClose}
className="w-full bg-gray-600 text-white font-bold py-3 px-4 rounded-lg"
>
Cancel
</button>
<button
type="submit"
disabled={pan.length !== 10 || aadhaar.length !== 12}
className="w-full bg-emerald-600 text-white font-bold py-3 px-4 rounded-lg
disabled:bg-gray-500"
>
Submit
</button>
</div>
</form>
</div>
</div>
);
};

/**
* Notification Component
```

```
* Displays success or error messages.
*/
const Notification = ({ message, type, onClose }) => {
if (!message) return null;

const isSuccess = type === 'success';
const bgColor = isSuccess ? 'bg-emerald-600' : 'bg-red-600';
const Icon = isSuccess ? CheckCircle : XCircle;

return (
<div className={`fixed top-5 left-1/2 -translate-x-1/2 z-50 w-full max-w-sm p-4 rounded-lg
shadow-lg text-white ${bgColor}`}>
<div className="flex items-center gap-3">
<Icon className="w-6 h-6" />
<span className="flex-grow">{message}</span>
<button onClick={onClose} className="text-white">&times;</button>
</div>
</div>
);
};

/**
* Loading Spinner Component
*/
const LoadingSpinner = () => (
<div className="flex flex-col items-center justify-center h-screen bg-gray-900 text-white">
<Loader2 className="w-12 h-12 animate-spin text-emerald-400" />
<p className="mt-4 text-lg">Loading Your CoinGrow Dashboard...</p>
</div>
);

/**
* Main App Component
* Manages state, Firebase logic, and renders views.
*/
export default function App() {
const [currentView, setView] = useState('dashboard');
const [firebaseState, setFirebaseState] = useState({
auth: null,
db: null,
userId: null,
isAuthReady: false
```

```
});

// User data
const [userProfile, setUserProfile] = useState(null);
const [activePlans, setActivePlans] = useState([]);
const [activeVipPlans, setActiveVipPlans] = useState([]);

// UI state
const [loading, setLoading] = useState(true);
const [notification, setNotification] = useState({ message: '', type: '' });
const [showKycModal, setShowKycModal] = useState(false);

// Ref to prevent multiple initializations
const firebaseInitialized = useRef(false);

// --- 1. Firebase Initialization and Auth ---
useEffect(() => {
if (firebaseInitialized.current || !auth || !db) return;
firebaseInitialized.current = true;
console.log("Firebase starting...");

const unsubscribe = onAuthStateChanged(auth, async (user) => {
if (user) {
console.log("User is signed in:", user.uid);
setFirebaseState({ auth, db, userId: user.uid, isAuthReady: true });
} else {
console.log("No user. Signing in...");
try {
if (typeof initial_auth_token !== 'undefined' && initial_auth_token) {
console.log("Signing in with custom token...");
await signInWithCustomToken(auth, initial_auth_token);
} else {
console.log("Signing in anonymously...");
await signInAnonymously(auth);
}
} catch (error) {
console.error("Auth error:", error);
showNotification(`Auth error: ${error.message}`, 'error');
}
}
});
return () => unsubscribe();
```

```
}, []);

// --- 2. Data Listeners (Firestore) ---
useEffect(() => {
if (!firebaseState.isAuthReady || !firebaseState.userId) return;

setLoading(true);
const { userId } = firebaseState;
const userDocPath = `artifacts/${appId}/users/${userId}/userData/profile`;
const plansCollectionPath = `artifacts/${appId}/users/${userId}/activePlans`;
const vipPlansCollectionPath = `artifacts/${appId}/users/${userId}/activeVipPlans`;

// --- User Profile Listener ---
const unsubProfile = onSnapshot(doc(db, userDocPath),
async (docSnap) => {
if (docSnap.exists()) {
console.log("User profile updated:", docSnap.data());
setUserProfile(docSnap.data());
} else {
console.log("No user profile. Creating one...");
// Create initial user profile
const initialProfile = {
totalBalance: 0,
availableBalance: 0,
totalEarnings: 0,
totalDeposits: 0,
kycStatus: 'none', // 'none', 'pending', 'verified'
createdAt: Timestamp.now(),
};
try {
await setDoc(doc(db, userDocPath), initialProfile);
setUserProfile(initialProfile);
} catch (e) {
console.error("Error creating user profile:", e);
}
}
setLoading(false);
}, (error) => {
console.error("Profile listener error:", error);
showNotification("Error loading profile.", 'error');
setLoading(false);
}
```

```
);

// --- Active Core Plans Listener ---
const unsubPlans = onSnapshot(collection(db, plansCollectionPath),
(snapshot) => {
const plans = snapshot.docs.map(doc => ({ id: doc.id, ...doc.data() }));
console.log("Active core plans updated:", plans);
setActivePlans(plans);
}, (error) => {
console.error("Core plans listener error:", error);
}
);

// --- Active VIP Plans Listener ---
const unsubVipPlans = onSnapshot(collection(db, vipPlansCollectionPath),
(snapshot) => {
const plans = snapshot.docs.map(doc => ({ id: doc.id, ...doc.data() }));
console.log("Active VIP plans updated:", plans);
setActiveVipPlans(plans);
}, (error) => {
console.error("VIP plans listener error:", error);
}
);

return () => {
unsubProfile();
unsubPlans();
unsubVipPlans();
};

}, [firebaseState.isAuthReady, firebaseState.userId]);

// --- 3. Simulation Engine (Earnings & Plan Maturity) ---
useEffect(() => {
if (!firebaseState.isAuthReady || !userProfile) return;

console.log("Starting simulation engine...");
const engineInterval = setInterval(async () => {
console.log("Engine tick: Calculating earnings...");
const { userId } = firebaseState;
const userRef = doc(db, `artifacts/${appId}/users/${userId}/userData/profile`);
```

```
let dailyProfit = 0;
activePlans.forEach(plan => {
dailyProfit += plan.dailyEarning;
});

const batch = writeBatch(db);

// Add daily earnings
if (dailyProfit > 0) {
batch.update(userRef, {
totalBalance: increment(dailyProfit),
availableBalance: increment(dailyProfit),
totalEarnings: increment(dailyProfit)
});
}

// Check for matured VIP plans
const now = Timestamp.now();
const maturedVipPlans = activeVipPlans.filter(
p => !p.isMatured && p.endDate.toMillis() <= now.toMillis()
);

if (maturedVipPlans.length > 0) {
console.log(`Processing ${maturedVipPlans.length} matured VIP plans...`);
let totalVipReturn = 0;

maturedVipPlans.forEach(plan => {
totalVipReturn += plan.totalReturn;
const planRef = doc(db, `artifacts/${appId}/users/${userId}/activeVipPlans`, plan.id);
batch.update(planRef, { isMatured: true });
});

// Add total return to balance
batch.update(userRef, {
totalBalance: increment(totalVipReturn),
availableBalance: increment(totalVipReturn)
});
}

// Commit batch if anything changed
if (dailyProfit > 0 || maturedVipPlans.length > 0) {
try {
```

```
await batch.commit();
console.log("Engine tick: Batch committed.");
if(maturedVipPlans.length > 0) showNotification("A VIP plan has matured! Check your balance.", 'success');
} catch (e) {
console.error("Engine tick batch commit error:", e);
}
} else {
console.log("Engine tick: No changes.");
}

}, 10000); // Runs every 10 seconds to simulate 1 day

return () => clearInterval(engineInterval);

}, [firebaseState.isAuthReady, userProfile, activePlans, activeVipPlans]);

// --- UI Helper Functions ---

const showNotification = (message, type) => {
setNotification({ message, type });
setTimeout(() => {
setNotification({ message: '', type: '' });
}, 3000);
};

// --- Action Handlers (Firestore Writes) ---

const handleDeposit = async (amount) => {
if (amount < MIN_DEPOSIT) {
showNotification(`Minimum deposit is ${formatCurrency(MIN_DEPOSIT)}`, 'error');
return;
}

setLoading(true);
const userRef = doc(db, `artifacts/${appId}/users/${firebaseState.userId}/userData/profile`);
try {
// In a real app, this would be after a payment gateway success.
await updateDoc(userRef, {
totalBalance: increment(amount),
availableBalance: increment(amount),
totalDeposits: increment(amount)
```

```
});
showNotification(`Successfully deposited ${formatCurrency(amount)}`, 'success');
} catch (e) {
console.error("Deposit error:", e);
showNotification('Deposit failed. Please try again.', 'error');
}
setLoading(false);
};

const handleWithdraw = async (amount) => {
if (userProfile.kycStatus !== 'verified') {
showNotification('Please complete KYC to withdraw.', 'error');
setShowKycModal(true);
return;
}
if (amount < MIN_WITHDRAWAL) {
showNotification(`Minimum withdrawal is ${formatCurrency(MIN_WITHDRAWAL)}`, 'error');
return;
}
if (amount > MAX_WITHDRAWAL) {
showNotification(`Maximum withdrawal is ${formatCurrency(MAX_WITHDRAWAL)}`, 'error');
return;
}
if (amount > userProfile.availableBalance) {
showNotification('Insufficient balance.', 'error');
return;
}

setLoading(true);
const userRef = doc(db, `artifacts/${appId}/users/${firebaseState.userId}/userData/profile`);
try {
await updateDoc(userRef, {
totalBalance: increment(-amount),
availableBalance: increment(-amount)
});
showNotification(`Withdrawal of ${formatCurrency(amount)} processed.`, 'success');
} catch (e) {
console.error("Withdrawal error:", e);
showNotification('Withdrawal failed. Please try again.', 'error');
}
setLoading(false);
};
```

```
const handleInvest = async (plan) => {
if (plan.investment > userProfile.availableBalance) {
showNotification('Insufficient balance to invest.', 'error');
return;
}

setLoading(true);
const { userId } = firebaseState;
const userRef = doc(db, `artifacts/${appId}/users/${userId}/userData/profile`);
const plansCollectionRef = collection(db, `artifacts/${appId}/users/${userId}/activePlans`);

try {
const batch = writeBatch(db);

// 1. Add new plan
batch.set(doc(plansCollectionRef), {
planId: plan.id,
planName: plan.name,
investment: plan.investment,
dailyEarning: plan.dailyEarning,
startDate: Timestamp.now()
});

// 2. Update user balance
batch.update(userRef, {
availableBalance: increment(-plan.investment)
});

await batch.commit();
showNotification(`Successfully invested in ${plan.name}!`, 'success');
setView('dashboard'); // Go to dashboard to see active plan
} catch (e) {
console.error("Investment error:", e);
showNotification('Investment failed. Please try again.', 'error');
}
setLoading(false);
};

const handleVipInvest = async (plan) => {
const vipInfo = getVipInfo(userProfile.totalDeposits);
if (plan.level > vipInfo.level) {
```

```
showNotification(`Requires VIP ${plan.level} to invest.`, 'error');
return;
}
if (plan.investment > userProfile.availableBalance) {
showNotification('Insufficient balance to invest.', 'error');
return;
}

setLoading(true);
const { userId } = firebaseState;
const userRef = doc(db, `artifacts/${appId}/users/${userId}/userData/profile`);
const vipPlansCollectionRef = collection(db,
`artifacts/${appId}/users/${userId}/activeVipPlans`);

try {
const batch = writeBatch(db);
const startDate = Timestamp.now();
const endDate = Timestamp.fromMillis(startDate.toMillis() + plan.termDays  24  60  60  1000);

// 1. Add new VIP plan
batch.set(doc(vipPlansCollectionRef), {
planId: plan.id,
planName: plan.name,
investment: plan.investment,
totalReturn: plan.totalReturn,
termDays: plan.termDays,
startDate: startDate,
endDate: endDate,
isMatured: false
});

// 2. Update user balance
batch.update(userRef, {
availableBalance: increment(-plan.investment)
});

await batch.commit();
showNotification(`Successfully invested in ${plan.name}!`, 'success');
setView('dashboard'); // Go to dashboard to see active plan
} catch (e) {
console.error("VIP Investment error:", e);
showNotification('Investment failed. Please try again.', 'error');
```

```
}
setLoading(false);
};

const handleKycSubmit = async (pan, aadhaar) => {
setLoading(true);
const userRef = doc(db, `artifacts/${appId}/users/${firebaseState.userId}/userData/profile`);
try {
// Simulate submission
await updateDoc(userRef, { kycStatus: 'pending' });

// Simulate admin approval after 3 seconds
setTimeout(async () => {
try {
await updateDoc(userRef, { kycStatus: 'verified' });
showNotification('KYC Verified! You can now withdraw.', 'success');
} catch (e) {
console.error("KYC verification error:", e);
}
}, 3000);

setShowKycModal(false);
showNotification('KYC submitted for review.', 'success');
} catch (e) {
console.error("KYC submission error:", e);
showNotification('KYC submission failed.', 'error');
}
setLoading(false);
};

// --- Render Logic ---

if (loading || !userProfile) {
return <LoadingSpinner />;
}

const vipInfo = getVipInfo(userProfile.totalDeposits);

const renderView = () => {
switch (currentView) {
case 'dashboard':
return <Dashboard
```

```
userProfile={userProfile}
activePlans={activePlans}
activeVipPlans={activeVipPlans}
vipInfo={vipInfo}
/>;
case 'invest':
return <Invest onInvest={handleInvest} userBalance={userProfile.availableBalance} />;
case 'vip':
return <Vip onVipInvest={handleVipInvest} userBalance={userProfile.availableBalance}
vipInfo={vipInfo} />;
case 'wallet':
return <Wallet
userProfile={userProfile}
onDeposit={handleDeposit}
onWithdraw={handleWithdraw}
onKycSubmit={handleKycSubmit}
setShowKycModal={setShowKycModal}
/>;
default:
return <Dashboard
userProfile={userProfile}
activePlans={activePlans}
activeVipPlans={activeVipPlans}
vipInfo={vipInfo}
/>;
}
};

return (
<div className="flex flex-col h-screen bg-gray-900 text-white font-sans">
<Notification
message={notification.message}
type={notification.type}
onClose={() => setNotification({ message: '', type: '' })}
/>
{showKycModal && (
<KycModal
onClose={() => setShowKycModal(false)}
onSubmit={handleKycSubmit}
/>
)}
```

```
<Header userProfile={userProfile} vipInfo={vipInfo} />

<main className="flex-grow overflow-y-auto">
{renderView()}
</main>

<BottomNav currentView={currentView} setView={setView} />
</div>
);
}
```