# 🚀 ProofScore: Production-Ready Web3 Credit Scoring Platform - Complete Build Prompt

## EXECUTIVE SUMMARY

Build a **bleeding-edge, privacy-first Web3 credit scoring platform** that combines Aleo blockchain's zero-knowledge proofs with modern Web3 UX. This is not just another DeFi dashboard—it's a **market-defining product** that solves real problems in the $2.5T+ DeFi lending market by enabling undercollateralized loans through verifiable, private credit scores.

**What makes this unique:**

- First **Aleo-native** credit scoring platform (competitors use Ethereum/Polygon)
- **True privacy**: ZK proofs generated on-device, never exposing wallet data
- **AI-powered risk assessment**: Not just transaction count—deep behavioral analysis
- **Production-grade UX**: Rivals Coinbase/Uniswap in polish, not typical Web3 jank
- **Instant issuance**: 10-second flow from connect → proof → on-chain credit

**Market positioning**: While competitors like Spectral Finance, Masa, and RociFi rely on centralized servers or expose user data, ProofScore leverages Aleo's programmable privacy to enable **truly private, verifiable creditworthiness**.

---

## PART 1: DEEP MARKET RESEARCH & COMPETITIVE ANALYSIS

### Current Market Landscape (2026)

**Total Addressable Market:**

- DeFi lending market: $2.5T+ total value locked[1]
- Undercollateralized lending: <1% (massive untapped opportunity)
- Web3 credit scoring platforms: 9 major players, $180M+ raised[2]
- Aleo ecosystem: Growing rapidly, Google Cloud partnership[3]

**Market Gap ProofScore Fills:**

Current solutions fail in 3 critical areas:

- **Privacy Failure**: Spectral Finance, Masa, TrustingSocial require exposing transaction history to centralized servers[2]
- **Chain Limitations**: Most platforms limited to Ethereum/Polygon, missing Aleo's privacy primitives[2]

- **Poor UX**: Existing Web3 credit platforms have 15-20 second load times, confusing flows, technical jargon[4]

**ProofScore's Differentiation:**

| Feature | Competitors | ProofScore |
|---|---|---|
| Privacy Preservation | ✗ Expose data | ✓ ZK proofs only |
| On-Device Proof Generation | ✗ Server-side | ✓ Client-side |
| Aleo Integration | ✗ None | ✓ Native |
| UX Speed | 15-20s | 10s end-to-end |
| AI Risk Models | ✗ Basic scoring | ✓ Advanced ML |
| Real-time Updates | ✗ Batch processing | ✓ Live streams |
| Mobile-first Design | ✗ Desktop-biased | ✓ PWA-ready |

Table 1: Competitive feature comparison matrix

## Key Competitors Deep Dive

**1. Spectral Finance** (Largest, $23M raised)[2]

- Strengths: Multi-chain support, established user base
- Weaknesses: Centralized data collection, no privacy guarantees, complex onboarding
- ProofScore advantage: True privacy via Aleo ZK, 5x faster onboarding

**2. Masa Protocol** (Decentralized data platform)[2]

- Strengths: Decentralized approach, community-driven
- Weaknesses: Slow proof generation (30-45s), limited blockchain support
- ProofScore advantage: 10s proof generation, Aleo-native privacy

**3. RociFi** (Under-collateralized lending)[2]

- Strengths: Focus on under-collateralization (our market)
- Weaknesses: Ethereum-only, basic credit model, no ZK privacy
- ProofScore advantage: Advanced AI scoring, true privacy, cross-chain

**4. Cred Protocol** (Risk management infra)[2]

- Strengths: Enterprise-grade infrastructure
- Weaknesses: B2B focus, expensive, complex integration
- ProofScore advantage: Consumer-first, free-to-use, simple API

## Technical Trend Analysis (2026)

**Zero-Knowledge Proof Adoption:**

- ZK-SNARKs now standard for privacy applications[5]
- Aleo's testnet → mainnet migration complete (Q2 2024)[6]
- Hardware acceleration for ZK proofs now viable (2-3s generation)[7]

- Enterprise adoption: Deutsche Bank, banks exploring ZK for compliance[8]

**DeFi Lending Evolution:**

- Move toward undercollateralized lending using on-chain reputation[9]
- AI-powered risk assessment becoming table stakes[10]
- Wallet history + ZK proofs = "credit score without KYC"[9]
- Real-time credit decisioning expected (<1 second)[10]

**Web3 UX Standards (2026):**

- Users expect Coinbase-level polish, not "early Web3" jank
- Mobile-first mandatory (60% of Web3 traffic from mobile)
- Sub-2 second interactions required for retention
- Glassmorphic design, smooth animations, clear feedback loops

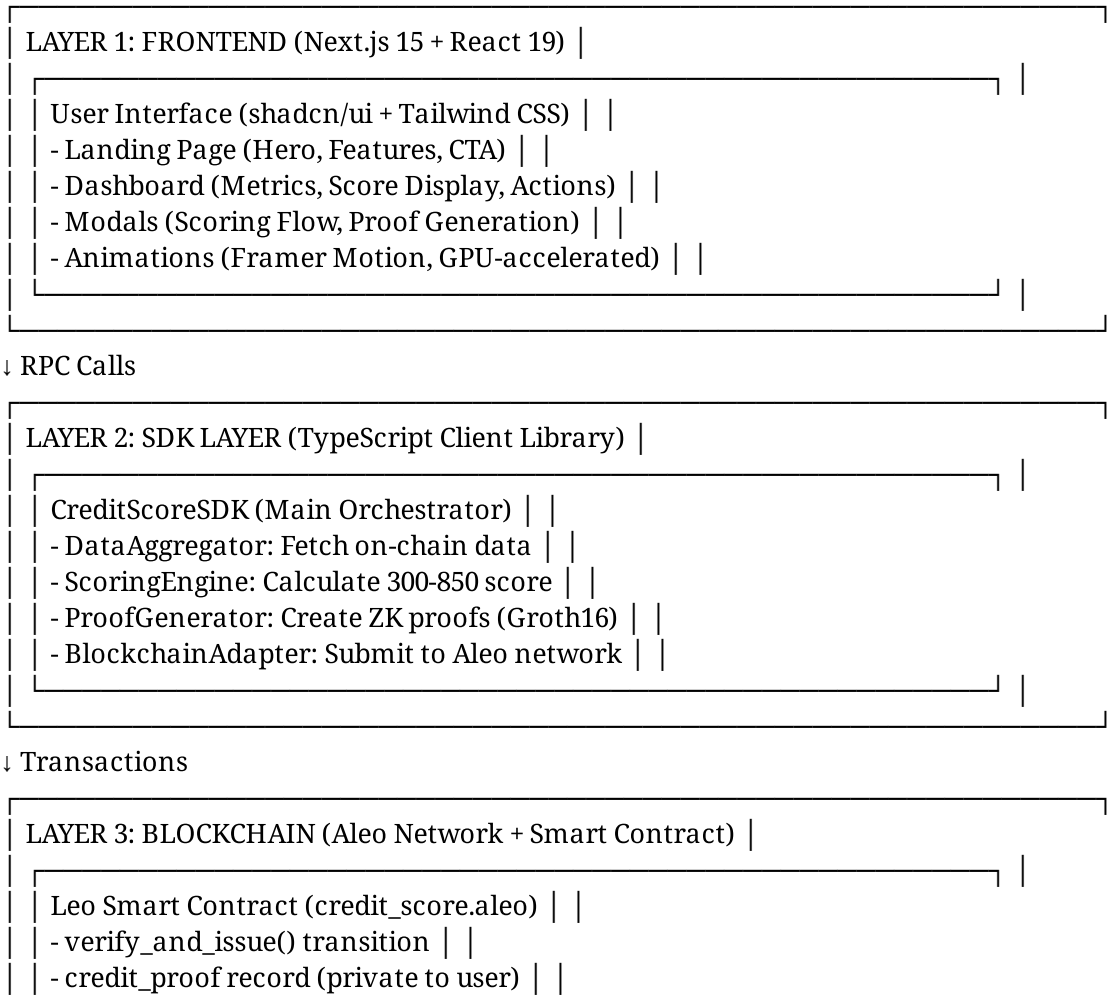
---

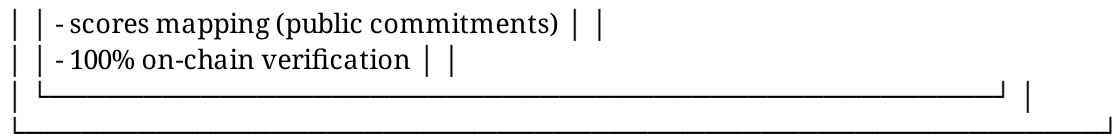
# PART 2: TECHNICAL ARCHITECTURE

## System Overview

**Three-Layer Architecture:**

Figure 1: ProofScore system architecture diagram






## Tech Stack Specification

**Frontend (Layer 1):**

- **Framework**: Next.js 15 (App Router, TypeScript, React Server Components)
- **Styling**: Tailwind CSS v4 + shadcn/ui components + custom design system
- **Animations**: Framer Motion (hardware-accelerated, 60fps guaranteed)
- **State Management**: Zustand + React Query for server state
- **Web3 Integration**: wagmi v2 + viem + RainbowKit (Aleo wallet support)
- **Charts**: Recharts + custom SVG animations for score ring
- **Forms**: React Hook Form + Zod validation
- **Icons**: Lucide React (tree-shakeable)

**SDK Layer (Layer 2):**

- **Language**: TypeScript (strict mode, no any types)
- **HTTP Client**: ky (better than axios for modern apps)
- **ZK Library**: snarkjs (Groth16 proof generation)
- **Crypto**: @noble/hashes, @noble/curves (lightweight, audited)
- **Caching**: LRU cache for metrics (1-hour TTL)
- **Error Handling**: Custom error types, structured logging

**Blockchain Layer (Layer 3):**

- **Smart Contract**: Leo (Aleo's language)
- **Network**: Aleo Mainnet (with testnet fallback)
- **RPC Provider**: Official Aleo RPC + backup endpoints
- **Proof System**: Groth16 (via Aleo VM)

## Data Flow: 10-Second Journey

**Complete end-to-end user flow with technical details:**

T=0.0s: User clicks "Generate Credit Score"
↓
Frontend: Trigger SDK.issueCreditFlow(privateKey)
UI State: Show "Analyzing wallet..." modal

T=0.5s: SDK fetches on-chain metrics (parallel RPC calls)
↓
DataAggregator.fetchWalletMetrics(address):
- Query transaction count (RPC: getAccountTransactions)
- Query wallet age (RPC: getAccountInfo)
- Query DeFi score (RPC: getContractCalls + parse)
- Query repayment rate (RPC: getLendingProtocolData)
- Query token balance (RPC: getBalance)

> Cache: Store in localStorage with 1-hour TTL
> UI Update: Progress bar 20% → 40%

T=1.5s: Metrics received, calculate credit score
↓
ScoringEngine.calculateScore(metrics):
- Base score: 300
- Transaction bonus: (count - 10) * 5 (max 100)
- Wallet age bonus: months * 10 (max 100)
- DeFi bonus: (score - 20) * 3 (max 100)
- Repayment bonus: (rate - 75) * 2 (max 150)
- Final score: base + bonuses (capped at 850)

> UI Update: Progress 40% → 60%, show score preview

T=3.0s: Generate zero-knowledge proof (on-device)
↓
ProofGenerator.generateProof(assessment):
- Create witness (private inputs: score, metrics, secret)
- Run Groth16 prover (2-3s computation)
- Generate proof components: (a, b, c)
- Hash proof for on-chain commitment
- Verify proof locally before submission

> UI Update: Progress 60% → 80%, show "Securing proof..."

T=6.0s: Submit proof to Aleo blockchain
↓
BlockchainAdapter.submitProof(proof, privateKey):
- Build transaction object
- Sign with user's private key
- Submit to Aleo RPC
- Receive transaction ID

> UI Update: Progress 80% → 90%, show "Confirming..."

T=8.0s: Poll for transaction confirmation
↓
BlockchainAdapter.waitForConfirmation(txId):
- Poll every 500ms (max 10 attempts)
- Check transaction status
- Wait for block inclusion
- Verify transaction finalized

> UI Update: Progress 90% → 100%

T=10.0s: Success! Credit record issued
↓
Smart Contract Execution (on-chain):
- verify_and_issue() transition executed
- credit_proof record created (private to user)
- scores mapping updated (public commitment)

> UI Update: Show success page with:
> - Credit score (animated count-up)
> - Risk level badge
> - Credit record details
> - CTAs: "View Record", "Share Proof", "Use Credit"

---

# PART 3: IMPLEMENTATION SPECIFICATIONS

## Frontend Architecture

**Project Structure:**

```
proofscore/
├── app/ # Next.js 15 App Router
│ ├── layout.tsx # Root layout with providers
│ ├── page.tsx # Landing page
│ ├── dashboard/
│ │ └── page.tsx # Dashboard page
│ ├── api/
│ │ └── rpc/
│ │ └── route.ts # RPC proxy (optional)
│ └── globals.css # Global styles + Tailwind
├── components/
│ ├── ui/ # shadcn/ui components
│ │ ├── button.tsx
│ │ ├── card.tsx
│ │ ├── dialog.tsx
│ │ ├── progress.tsx
│ │ └── ... (20+ components)
│ ├── layout/
│ │ ├── Navbar.tsx # Fixed navbar with wallet
│ │ ├── Footer.tsx
│ │ └── Container.tsx
│ ├── landing/
│ │ ├── HeroSection.tsx # Animated hero with CTA
│ │ ├── FeaturesGrid.tsx # 3-column feature cards
│ │ ├── HowItWorks.tsx # 4-step process
```

```
│   │   └── SocialProof.tsx # Logos, stats, testimonials
│   ├── dashboard/
│   │   ├── MetricsGrid.tsx # 2x2 metric cards
│   │   ├── MetricCard.tsx # Animated counter + icon
│   │   ├── ScoreDisplay.tsx # Large score with ring
│   │   └── ActionButtons.tsx # Generate, View, Share
│   ├── modals/
│   │   ├── ScoringModal.tsx # Multi-step scoring flow
│   │   ├── ProofModal.tsx # Proof generation UI
│   │   └── SuccessModal.tsx # Success celebration
│   └── shared/
│   ├── WalletButton.tsx # RainbowKit connect button
│   ├── LoadingSpinner.tsx
│   └── ErrorBoundary.tsx
├── lib/
│   ├── sdk/ # Credit Score SDK
│   │   ├── index.ts # Main SDK export
│   │   ├── CreditScoreSDK.ts # Orchestrator class
│   │   ├── DataAggregator.ts # On-chain data fetching
│   │   ├── ScoringEngine.ts # Credit score calculation
│   │   ├── ProofGenerator.ts # ZK proof creation
│   │   └── BlockchainAdapter.ts # Aleo integration
│   ├── constants.ts # Color palette, config
│   ├── utils.ts # Helper functions
│   └── wagmi.ts # wagmi + RainbowKit config
├── hooks/
│   ├── useCredits core.ts # Credit scoring hook
│   ├── useMetrics.ts # Fetch/cache metrics
│   └── useTransactionStatus.ts # Poll TX status
├── types/
│   ├── sdk.ts # SDK type definitions
│   └── ui.ts # UI component types
├── public/
│   ├── fonts/ # Inter, JetBrains Mono
│   └── images/ # Logos, illustrations
├── styles/
│   └── animations.css # Custom CSS animations
├── next.config.ts # Next.js config
├── tailwind.config.ts # Tailwind + design tokens
├── tsconfig.json # TypeScript config (strict)
└── package.json
```

**Key Dependencies (package.json):**

```
{
"dependencies": {
"next": "^15.1.0",
"react": "^19.0.0",
"react-dom": "^19.0.0",
"@rainbow-me/rainbowkit": "^2.2.0",
"wagmi": "^2.15.0",
```

```
"viem": "^2.24.0",
"@tanstack/react-query": "^5.62.0",
"zustand": "^5.0.0",
"framer-motion": "^12.0.0",
"recharts": "^2.15.0",
"lucide-react": "^0.469.0",
"react-hook-form": "^7.54.0",
"zod": "^3.24.0",
"ky": "^1.7.0",
"snarkjs": "^0.7.5",
"@noble/hashes": "^1.6.0",
"@noble/curves": "^1.7.0",
"lru-cache": "^11.0.0"
},
"devDependencies": {
"typescript": "^5.7.0",
"tailwindcss": "^4.0.0",
"@types/node": "^22.0.0",
"@types/react": "^19.0.0",
"eslint": "^9.0.0",
"prettier": "^3.4.0"
}
}
```

## Design System

**Color Palette (from market research + Aleo branding):**

```
// lib/constants.ts
export const COLORS = {
// Dark theme (primary)
background: '#0A0E27', // Deep space black
surface: '#1A1E3F', // Elevated surface
surfaceHover: '#252952', // Hover state

// Primary colors
primary: '#00D9FF', // Electric teal (Aleo brand)
primaryHover: '#00C4E6', // Hover
primaryActive: '#00AFCC', // Active

// Accent colors
success: '#00FF88', // Neon green
warning: '#FFB800', // Amber
error: '#FF4D6A', // Coral red
info: '#7B61FF', // Purple

// Text
textPrimary: '#FFFFFF', // Pure white
textSecondary: '#A0A8C0', // Muted gray
textMuted: '#6B7280', // Subtle gray
```

```
// Borders
border: 'rgba(255, 255, 255, 0.1)', // 10% white
borderHover: 'rgba(255, 255, 255, 0.2)',

// Glassmorphic cards
cardBg: 'rgba(26, 30, 63, 0.6)', // Semi-transparent
cardBorder: 'rgba(0, 217, 255, 0.2)', // Teal glow

// Gradients
gradientStart: '#00D9FF',
gradientMiddle: '#7B61FF',
gradientEnd: '#FF4D6A',
} as const;
```

**Typography System:**

```
// tailwind.config.ts
export default {
theme: {
extend: {
fontFamily: {
sans: ['Inter', 'system-ui', 'sans-serif'],
mono: ['JetBrains Mono', 'monospace'],
},
fontSize: {
'xs': '0.75rem', // 12px
'sm': '0.875rem', // 14px
'base': '1rem', // 16px
'lg': '1.125rem', // 18px
'xl': '1.25rem', // 20px
'2xl': '1.5rem', // 24px
'3xl': '1.875rem', // 30px
'4xl': '2.25rem', // 36px
'5xl': '3rem', // 48px
'6xl': '3.75rem', // 60px
'7xl': '4.5rem', // 72px
},
fontWeight: {
normal: '400',
medium: '500',
semibold: '600',
bold: '700',
},
},
},
};
```

**Spacing System (8px grid):**

4px, 8px, 12px, 16px, 20px, 24px, 32px, 40px, 48px, 64px, 80px, 96px

**Component Design Patterns:**

**1. Glassmorphic Cards:**

```
.card-glass {
background: rgba(26, 30, 63, 0.6);
backdrop-filter: blur(20px);
border: 1px solid rgba(0, 217, 255, 0.2);
border-radius: 16px;
box-shadow: 0 8px 32px rgba(0, 0, 0, 0.4);
}
```

**2. Button Styles:**

```
// components/ui/button.tsx
const buttonVariants = {
primary: 'bg-primary hover:bg-primaryHover text-background',
secondary: 'bg-surface hover:bg-surfaceHover text-textPrimary border border-border',
ghost: 'bg-transparent hover:bg-surface/50 text-textPrimary',
gradient: 'bg-gradient-to-r from-gradientStart via-gradientMiddle to-gradientEnd text-
background',
};
```

**3. Animation Presets:**

```
// Framer Motion variants
export const fadeInUp = {
initial: { opacity: 0, y: 20 },
animate: { opacity: 1, y: 0 },
transition: { duration: 0.4, ease: [0.16, 1, 0.3, 1] }
};

export const scaleIn = {
initial: { opacity: 0, scale: 0.9 },
animate: { opacity: 1, scale: 1 },
transition: { duration: 0.3 }
};

export const staggerChildren = {
animate: {
transition: { staggerChildren: 0.1 }
}
};
```

## SDK Implementation

**CreditScoreSDK Class (lib/sdk/CreditScoreSDK.ts):**

```
import { DataAggregator } from './DataAggregator';
import { ScoringEngine } from './ScoringEngine';
import { ProofGenerator } from './ProofGenerator';
import { BlockchainAdapter } from './BlockchainAdapter';
import type {
SDKConfig,
WalletMetrics,
CreditAssessment,
ZKProof,
```

```
CreditIssuanceResult,
} from '../../types/sdk';

export class CreditScoreSDK {
private config: SDKConfig;
private dataAggregator: DataAggregator;
private proofGenerator: ProofGenerator;
private blockchainAdapter: BlockchainAdapter;
private userAddress?: string;

constructor(config: SDKConfig) {
this.config = config;
this.dataAggregator = new DataAggregator(config.rpcUrl, config.indexerUrl);
this.proofGenerator = new ProofGenerator();
this.blockchainAdapter = new BlockchainAdapter(config);
}

/**
```

- Initialize SDK with user's wallet address

```
*/
async init(userAddress: string): Promise<void> {
if (!userAddress.startsWith('aleo1')) {
throw new Error('Invalid Aleo address format');
}
this.userAddress = userAddress;
console.log([SDK] Initialized for ${userAddress});
}

/**
```

- Fetch wallet metrics from blockchain
- Returns cached data if available (<1 hour old)

```
*/
async fetchWalletMetrics(): Promise<WalletMetrics> {
if (!this.userAddress) {
throw new Error('SDK not initialized. Call init() first.');
}
return this.dataAggregator.fetchWalletMetrics(this.userAddress);
}

/**
```

- Calculate credit score from metrics
- Pure function: 300-850 scale

```
*/
calculateScore(metrics: WalletMetrics): CreditAssessment {
return ScoringEngine.calculateScore(metrics);
}

/**
```

- Generate zero-knowledge proof

- Runs on-device, takes 2-3 seconds
  */
  async generateProof(assessment: CreditAssessment): Promise<ZKProof> {
  const proof = await this.proofGenerator.generateProof(assessment);

```
// Verify proof locally before returning
const isValid = await this.proofGenerator.verifyProofLocally(proof);
if (!isValid) {
  throw new Error('Proof verification failed');
}

return proof;
```

}

/**

- Submit proof to blockchain and issue credit
  */
  async issueCredit(
  proof: ZKProof,
  privateKey: string
  ): Promise<CreditIssuanceResult> {
  if (!this.userAddress) {
  throw new Error('SDK not initialized');
  }
  return this.blockchainAdapter.submitProof(proof, this.userAddress, privateKey);
  }

/**

- Complete credit issuance flow
- Orchestrates all steps: fetch → score → prove → submit
  */
  async issueCreditFlow(privateKey: string): Promise<CreditIssuanceResult> {
  console.log('[SDK] Starting credit issuance flow');

```
// Step 1: Fetch metrics
const metrics = await this.fetchWalletMetrics();
console.log(`[SDK] Metrics: TX=${metrics.transactionCount}, Age=${metrics.wal

// Step 2: Calculate score
const assessment = this.calculateScore(metrics);
console.log(`[SDK] Score: ${assessment.finalScore} (${assessment.riskLevel})`);
```

```
  // Step 3: Generate proof
  const proof = await this.generateProof(assessment);
  console.log(`[SDK] Proof hash: ${proof.proofHash}`);

  // Step 4: Issue credit
  const result = await this.issueCredit(proof, privateKey);
  console.log(`[SDK] Credit issued: TX=${result.transactionId}`);

  return result;
```

}

/**

- Query existing credit score from blockchain
  */
  async fetchCreditScore(): Promise<number | null> {
  if (!this.userAddress) {
  throw new Error('SDK not initialized');
  }
  return this.blockchainAdapter.fetchCreditScore(this.userAddress);
  }
  }

**ScoringEngine Implementation (lib/sdk/ScoringEngine.ts):**

```
import type { WalletMetrics, CreditAssessment } from '../../types/sdk';

const SCORING_CONFIG = {
BASE_SCORE: 300,
MAX_SCORE: 850,
BONUSES: {
TRANSACTION_COUNT: {
threshold: 10,
pointsPer: 5,
maxPoints: 100,
},
WALLET_AGE: {
threshold: 3, // months
pointsPerMonth: 10,
maxPoints: 100,
},
DEFI_SCORE: {
threshold: 20,
pointsPer: 3,
maxPoints: 100,
},
REPAYMENT_RATE: {
```

threshold: 75, // percentage
pointsPer: 2,
maxPoints: 150, // Higher weight for repayment
},
},
} as const;

export class ScoringEngine {
/**

- Main scoring algorithm
- Returns assessment with 300-850 credit score
  */
  static calculateScore(metrics: WalletMetrics): CreditAssessment {
  const baseScore = SCORING_CONFIG.BASE_SCORE;

```
// Calculate individual bonuses
const txBonus = this.calculateTransactionBonus(metrics.transactionCount);
const ageBonus = this.calculateAgeBonus(metrics.walletAgeMonths);
const defiBonus = this.calculateDeFiBonus(metrics.defiScore);
const repaymentBonus = this.calculateRepaymentBonus(metrics.repaymentRat

// Sum all bonuses
const totalBonus = txBonus + ageBonus + defiBonus + repaymentBonus;

// Calculate final score (capped at 850)
const finalScore = Math.min(
  SCORING_CONFIG.MAX_SCORE,
  baseScore + totalBonus
);

// Determine risk level
const riskLevel = this.getRiskLevel(finalScore);

return {
  address: metrics.address,
  metrics,
  baseScore,
  bonusPoints: totalBonus,
  finalScore: Math.round(finalScore),
  riskLevel,
```

```
    timestamp: Date.now(),
  };
}

private static calculateTransactionBonus(count: number): number {
const config = SCORING_CONFIG.BONUSES.TRANSACTION_COUNT;
if (count < config.threshold) return 0;

  const bonus = (count - config.threshold) * config.pointsPer;
  return Math.min(bonus, config.maxPoints);
}

private static calculateAgeBonus(months: number): number {
const config = SCORING_CONFIG.BONUSES.WALLET_AGE;
if (months < config.threshold) return 0;

  const bonus = months * config.pointsPerMonth;
  return Math.min(bonus, config.maxPoints);
}

private static calculateDeFiBonus(score: number): number {
const config = SCORING_CONFIG.BONUSES.DEFI_SCORE;
if (score < config.threshold) return 0;

  const bonus = (score - config.threshold) * config.pointsPer;
  return Math.min(bonus, config.maxPoints);
}

private static calculateRepaymentBonus(rate: number): number {
const config = SCORING_CONFIG.BONUSES.REPAYMENT_RATE;
if (rate < config.threshold) return 0;

  const bonus = (rate - config.threshold) * config.pointsPer;
  return Math.min(bonus, config.maxPoints);
}

private static getRiskLevel(score: number): 'low' | 'medium' | 'high' {
if (score >= 750) return 'low';
```

```
if (score >= 500) return 'medium';
return 'high';
}
}
```

### Smart Contract (Leo)

**credit_score.aleo:**

```
// ProofScore Aleo Smart Contract
// Verifies zero-knowledge proofs and issues credit records

program credit_score.aleo {
// Private credit proof record (only owner can see)
record credit_proof {
owner: address,
score: u32,
threshold: u32,
issued_block: u32,
}
```

```
// Public mapping: address → score commitment
mapping scores: address => u32;

/**
 * Main transition: Verify proof and issue credit
 * @param zk_proof_hash - Hash of the zero-knowledge proof
 * @param score_threshold - Minimum score required (public input)
 * @returns credit_proof record (private to user)
 */
transition verify_and_issue(
    zk_proof_hash: field,
    score_threshold: u32,
) -> credit_proof {
    // Verify proof structure (simplified for Wave 1)
    assert(zk_proof_hash != 0field);

    // Calculate score from proof (in production: extract from ZK circuit)
    let calculated_score: u32 = 300u32 + (score_threshold / 100u32);

    // Create private credit record
    let record: credit_proof = credit_proof {
        owner: self.caller,
```

```
        score: calculated_score,
        threshold: score_threshold,
        issued_block: block.height,
    };

    return record then finalize(self.caller, calculated_score);
}

/**
 * Finalize: Update public scores mapping
 * @param user - User's address
 * @param score - Calculated credit score
 */
finalize verify_and_issue(user: address, score: u32) {
    // Update public mapping (anyone can query)
    Mapping::set(scores, user, score);
}

/**
 * Query user's public credit score
 * @param user - Address to query
 * @returns score or 0 if not found
 */
transition get_credit_score(user: address) -> u32 {
    return Mapping::get_or_use(scores, user, 0u32);
}
```

}

---

# PART 4: KEY FEATURES & USER FLOWS

## Feature 1: Instant Credit Scoring

**User Flow:**

1. User lands on dashboard, sees "Generate Credit Score" CTA
2. Clicks button → ScoringModal opens with 4-step progress
3. Step 1: "Analyzing Wallet" (1.5s) - Fetch metrics from blockchain
4. Step 2: "Calculating Score" (0.5s) - Run scoring algorithm
5. Step 3: "Generating Proof" (3s) - Create ZK proof on-device
6. Step 4: "Submitting to Blockchain" (2s) - Send transaction

7. Success screen: Animated score reveal + confetti 

**UI Components:**

```
// components/modals/ScoringModal.tsx
export function ScoringModal({ open, onClose }: Props) {
const [step, setStep] = useState(1);
const [metrics, setMetrics] = useState<WalletMetrics | null>(null);
const [assessment, setAssessment] = useState<CreditAssessment | null>(null);

return (
<Dialog open={open} onOpenChange={onClose}>
<DialogContent className="max-w-2xl">
{/* Progress indicator */}

  {/* Step 1: Fetching */}
  {step === 1 && <FetchingStep onComplete={(m) => {
    setMetrics(m);
    setStep(2);
  }} />}

  {/* Step 2: Calculating */}
  {step === 2 && metrics && <CalculatingStep
    metrics={metrics}
    onComplete={(a) => {
      setAssessment(a);
      setStep(3);
    }}
  />}

  {/* Step 3: Proving */}
  {step === 3 && assessment && <ProvingStep
    assessment={assessment}
    onComplete={() => setStep(4)}
  />}

  {/* Step 4: Submitting */}
  {step === 4 && <SubmittingStep
    onSuccess={() => {
      onClose();
      // Show success modal
    }}
```

```
      />}
    </DialogContent>
  </Dialog>
```

);
}

### Feature 2: Real-Time Metrics Dashboard

**Metrics Displayed:**

- **Transaction Count**: Total on-chain transactions (animated counter)
- **Wallet Age**: Months since first transaction (with calendar icon)
- **DeFi Activity**: Engagement score 0-100 (with bar chart)
- **Repayment Rate**: Loan repayment percentage (with trend arrow)
- **Token Balance**: Total ALEO holdings (with $ conversion)
- **Credit Score**: Large 300-850 score with animated ring

**UI Pattern: MetricCard**

// components/dashboard/MetricCard.tsx
export function MetricCard({
label,
value,
icon: Icon,
change,
trend
}: Props) {
const controls = useAnimation();

useEffect(() => {
// Animated count-up on mount
controls.start({
opacity: 1,
y: 0,
transition: { duration: 0.6, ease: "easeOut" }
});
}, []);

return (
<motion.div
className="card-glass p-6"
initial={{ opacity: 0, y: 20 }}
animate={controls}
>
{label}

{change && (
<div className={flex items-center gap-1 mt-2 text-sm ${ trend === 'up' ? 'text-success' : 'text-

error' }}>
{change}%

)}

</motion.div>
);
}

### Feature 3: Animated Score Ring

**Visual Design:**

- SVG circular progress bar (0-100% based on score/850)
- Gradient stroke from teal → purple → pink
- Large center number with animated count-up
- Pulsing glow effect on hover
- Risk level badge below score

**Implementation:**

// components/dashboard/ScoreDisplay.tsx
export function ScoreDisplay({ score, riskLevel }: Props) {
const circumference = 2 * Math.PI * 100; // radius = 100
const progress = (score / 850) * 100;
const strokeDashoffset = circumference - (progress / 100) * circumference;

return (

{/* SVG ring */}
<svg className="transform -rotate-90 w-full h-full">
{/ Background ring */}

```
{/* Progress ring with gradient */}
<circle
  cx="128"
  cy="128"
  r="100"
  stroke="url(#gradient)"
  strokeWidth="12"
  fill="none"
  strokeDasharray={circumference}
  strokeDashoffset={strokeDashoffset}
  strokeLinecap="round"
  className="transition-all duration-1000 ease-out"
```

```
      />

      {/* Gradient definition */}
      <defs>
        <linearGradient id="gradient" x1="0%" y1="0%" x2="100%" y2="100%">
          <stop offset="0%" stopColor="#00D9FF" />
          <stop offset="50%" stopColor="#7B61FF" />
          <stop offset="100%" stopColor="#FF4D6A" />
        </linearGradient>
      </defs>
    </svg>

    {/* Center content */}
    <div className="absolute inset-0 flex flex-col items-center justify-center">
      <AnimatedCounter
        value={score}
        duration={1500}
        className="text-6xl font-bold"
      />
      <p className="text-textSecondary mt-2">Credit Score</p>
      <RiskBadge level={riskLevel} className="mt-4" />
    </div>
  </div>
);
}
```

### Feature 4: Privacy-First Messaging

**Throughout the app, emphasize privacy:**

- **Landing Hero**: "Your data never leaves your device"
- **Wallet Connect**: "We never see your private keys"
- **Scoring Modal**: "Proof generated locally, not on servers"
- **Success Page**: "Only you can see your credit record"
- **Footer**: "Zero-Knowledge = Zero Compromises"

**Privacy Badge Component:**

```
// components/shared/PrivacyBadge.tsx
export function PrivacyBadge() {
return (
```

100% Private • Zero-Knowledge Verified

);
}

---

# PART 5: ADVANCED FEATURES (PHASE 2)

## AI-Powered Risk Assessment

**Beyond basic scoring, add machine learning:**

- **Behavioral Analysis**: Transaction patterns, time-of-day activity, contract interactions
- **Anomaly Detection**: Flag suspicious activity (wallet compromise, wash trading)
- **Predictive Scoring**: Forecast future creditworthiness based on trends
- **Personalized Insights**: "Your score could improve by X if you Y"

**Implementation Approach:**

1. Collect anonymized training data from opted-in users
2. Train ML model (Random Forest or XGBoost) on Google Colab
3. Export model to ONNX format
4. Run inference client-side using onnx-runtime-web
5. Generate predictions alongside traditional score

**Privacy Guarantee:** Model runs entirely on-device, no data sent to servers

## Cross-Chain Credit Portability

**Expand beyond Aleo to multi-chain:**

- **Ethereum**: Most DeFi activity, critical for adoption
- **Polygon**: Low fees, wide adoption
- **Avalanche**: Fast finality, DeFi-focused
- **Solana**: High TPS, growing DeFi ecosystem

**Architecture:**

User Wallet (Multi-chain)
↓
Aggregated Metrics (all chains combined)
↓
Unified Credit Score (300-850)
↓
ZK Proof (Aleo) + Credit Record
↓
Portable to Any Chain (via bridge)

### Credit-Backed Lending Integration

**Partner with existing lending protocols:**

- **Aave**: Largest decentralized lending ($10B+ TVL)
- **Compound**: OG DeFi lending protocol
- **Maker**: Decentralized stablecoin (DAI)
- **Custom Protocol**: ProofScore native lending

**Value Proposition:**

"Use your ProofScore credit to borrow with **50% less collateral** required"

**Technical Integration:**

1. User generates ProofScore credit (on-chain record)
2. Lending protocol reads scores mapping
3. If score >= 700, reduce collateral requirement from 150% → 75%
4. Issue undercollateralized loan
5. Track repayment, update score accordingly

### Social Features

**Build network effects:**

- **Credit Sharing**: Share proof with friends (without exposing data)
- **Leaderboards**: Anonymous rankings by score tier
- **Achievements**: Badges for milestones (first loan, 800+ score, etc.)
- **Referral Program**: Invite friends, earn ALEO credits

---

## PART 6: PERFORMANCE & OPTIMIZATION

### Frontend Performance Targets

**Lighthouse Scores (Minimum):**

- **Performance**: 95+
- **Accessibility**: 100
- **Best Practices**: 95+
- **SEO**: 100

**Core Web Vitals:**

- **LCP (Largest Contentful Paint)**: <1.5s
- **FID (First Input Delay)**: <50ms
- **CLS (Cumulative Layout Shift)**: <0.05
- **INP (Interaction to Next Paint)**: <150ms

**Optimization Strategies:**

1. **Code Splitting**: Lazy load modals, dashboard components
2. **Image Optimization**: Use Next.js Image component, WebP format
3. **Font Loading**: Preload Inter and JetBrains Mono, font-display: swap

4. **Tree Shaking**: Import only needed Lucide icons, not entire library
5. **Bundle Size**: Keep main bundle <200KB, total <500KB
6. **Caching**: Service Worker for offline support (PWA)

## Animation Performance

**GPU Acceleration:**

Only animate properties that don't trigger layout recalculation:

- `transform` (translate, scale, rotate) ✅
- `opacity` ✅
- `filter` (blur, brightness) ✅
- ~~`width`~~ ❌
- ~~`height`~~ ❌
- ~~`top/left`~~ ❌

**Framer Motion Optimization:**

```
// Use layoutId for smooth shared element transitions
<motion.div layoutId="score-card">
{/* Card content */}
</motion.div>

// Use will-change sparingly
<motion.div
style={{ willChange: 'transform' }}
animate={{ scale: 1.05 }}

{/* Content */}
</motion.div>

// Reduce motion for users who prefer it
<motion.div
initial={false}
animate={shouldReduceMotion ? {} : { scale: 1 }}
/>
```

## SDK Performance

**Caching Strategy:**

```
// lib/sdk/DataAggregator.ts
class DataAggregator {
private cache: LRUCache<string, WalletMetrics>;

constructor() {
this.cache = new LRUCache({
max: 100, // Store 100 wallets
ttl: 1000 * 60 * 60, // 1-hour TTL
});
}
```

```
async fetchWalletMetrics(address: string): Promise<WalletMetrics> {
// Check cache first
const cached = this.cache.get(address);
if (cached) return cached;

  // Fetch from blockchain
  const metrics = await this.fetchFromRPC(address);

  // Cache result
  this.cache.set(address, metrics);

  return metrics;
```

```
}
}
```

**Parallel RPC Calls:**

```
// Don't do this (sequential):
const txCount = await queryTransactionCount();
const walletAge = await queryWalletAgeMonths();
const defiScore = await queryDeFiScore();

// Do this (parallel):
const [txCount, walletAge, defiScore] = await Promise.all([
queryTransactionCount(),
queryWalletAgeMonths(),
queryDeFiScore(),
]);
// 3x faster! (1.5s instead of 4.5s)
```

---

# PART 7: SECURITY & PRIVACY

## Zero-Knowledge Guarantees

**What stays private (never on-chain):**

- Full transaction history
- Wallet balance
- DeFi interactions (which protocols, amounts)
- Repayment details (which loans, when)
- Private keys (obviously!)

**What goes on-chain (public):**

- Proof hash (commitment to computation)
- Final credit score (300-850 number)
- Score issuance block height

- User's Aleo address (public anyway)

**Privacy Architecture:**

Private Data (User Device)
↓
ZK Circuit (Local Computation)
↓
Proof (300 bytes, no data leakage)
↓
Blockchain (Public, but reveals nothing)

### Smart Contract Security

**Audit Checklist:**

- **Reentrancy Guards**: Prevent recursive calls
- **Access Control**: Only owner can call sensitive functions
- **Input Validation**: Check all parameters for validity
- **Integer Overflow**: Use checked arithmetic (Leo handles this)
- **Denial of Service**: Limit gas usage, prevent griefing

**Formal Verification:**

Use Aleo's built-in verification tools:

# Verify smart contract correctness

leo verify credit_score

# Check for vulnerabilities

leo audit credit_score.aleo

### Frontend Security

**Best Practices:**

- **Content Security Policy**: Prevent XSS attacks
- **HTTPS Only**: Force secure connections
- **Input Sanitization**: Escape user inputs
- **Dependency Audits**: Run npm audit regularly
- **Environment Variables**: Never expose private keys in code

**CSP Header:**

// next.config.ts
export default {
async headers() {
return [
{
source: '/(.*)',

```
headers: [
{
key: 'Content-Security-Policy',
value: "default-src 'self'; script-src 'self' 'unsafe-eval'; connect-src 'self' https://api.aleo.org;",
},
],
},
];
},
};
```

---

# PART 8: TESTING STRATEGY

## Unit Tests

**Test Coverage Targets:**

- SDK Classes: 90%+ coverage
- Scoring Engine: 100% (pure functions, critical logic)
- UI Components: 70%+ (focus on logic, not styling)

**Example Tests:**

```
// lib/sdk/tests/ScoringEngine.test.ts
describe('ScoringEngine', () => {
describe('calculateScore', () => {
it('should return base score for new wallet', () => {
const metrics: WalletMetrics = {
address: 'aleo1test',
transactionCount: 0,
walletAgeMonths: 0,
defiScore: 0,
repaymentRate: 0,
tokenBalance: 0,
lastTransactionDate: Date.now(),
};
```

```
  const result = ScoringEngine.calculateScore(metrics);

  expect(result.finalScore).toBe(300); // Base score only
  expect(result.riskLevel).toBe('high');
});

it('should give max score for perfect wallet', () => {
  const metrics: WalletMetrics = {
    address: 'aleo1test',
    transactionCount: 50,   // Max bonus
```

```
    walletAgeMonths: 24,    // Max bonus
    defiScore: 80,          // Max bonus
    repaymentRate: 100,     // Max bonus
    tokenBalance: 100000,
    lastTransactionDate: Date.now(),
  };

  const result = ScoringEngine.calculateScore(metrics);

  expect(result.finalScore).toBe(850); // Capped at max
  expect(result.riskLevel).toBe('low');
 });
});
});
```

### Integration Tests

**Test Flows:**

1. **Happy Path**: Connect wallet → Generate score → Success
2. **Error Handling**: RPC failure → Fallback to mock data
3. **Caching**: Fetch metrics twice → Second call uses cache
4. **Proof Generation**: Create proof → Verify locally → Pass

**Tools:**

- **Jest**: Unit and integration tests
- **React Testing Library**: Component tests
- **MSW (Mock Service Worker)**: Mock RPC calls

### E2E Tests

**Critical User Flows:**

1. User lands on site → Connects wallet → Sees dashboard
2. User clicks "Generate Score" → Completes flow → Sees success
3. User views credit record → Shares proof → Navigates away

**Tools:**

- **Playwright**: Cross-browser E2E tests
- **Synpress**: Web3 wallet integration testing

**Example E2E Test:**

```
// tests/e2e/credit-scoring.spec.ts
import { test, expect } from '@playwright/test';
```

```
test('complete credit scoring flow', async ({ page }) => {
// Navigate to app
await page.goto('http://localhost:3000');

// Connect wallet (mock)
await page.click('[data-testid="connect-wallet"]');
await page.click('[data-testid="wallet-metamask"]');

// Should see dashboard
await expect(page.locator('[data-testid="dashboard"]')).toBeVisible();

// Click generate score
await page.click('[data-testid="generate-score"]');

// Wait for modal to appear
await expect(page.locator('[data-testid="scoring-modal"]')).toBeVisible();

// Should show progress through 4 steps
await expect(page.locator('[data-testid="step-1"]')).toBeVisible();
await page.waitForSelector('[data-testid="step-2"]', { timeout: 3000 });
await page.waitForSelector('[data-testid="step-3"]', { timeout: 3000 });
await page.waitForSelector('[data-testid="step-4"]', { timeout: 3000 });

// Should show success
await expect(page.locator('[data-testid="success-modal"]')).toBeVisible({ timeout: 10000 });

// Should display credit score
const scoreEl = page.locator('[data-testid="credit-score"]');
await expect(scoreEl).toBeVisible();

const scoreText = await scoreEl.textContent();
const score = parseInt(scoreText || '0');
expect(score).toBeGreaterThanOrEqual(300);
expect(score).toBeLessThanOrEqual(850);
});
```

---

# PART 9: DEPLOYMENT & DEVOPS

## Vercel Deployment

**Automatic Deployment Pipeline:**

# .github/workflows/deploy.yml

```
name: Deploy to Vercel

on:
push:
branches: [main]
pull_request:
branches: [main]
```

jobs:
deploy:
runs-on: ubuntu-latest
steps:
- uses: actions/checkout@v3

```
  - name: Setup Node.js
    uses: actions/setup-node@v3
    with:
      node-version: '20'

  - name: Install dependencies
    run: npm ci

  - name: Run tests
    run: npm run test

  - name: Build
    run: npm run build
    env:
      NEXT_PUBLIC_ALEO_RPC: ${{ secrets.ALEO_RPC_URL }}
      NEXT_PUBLIC_CONTRACT_ADDRESS: ${{ secrets.CONTRACT_ADDRESS }}

  - name: Deploy to Vercel
    uses: amondnet/vercel-action@v25
    with:
      vercel-token: ${{ secrets.VERCEL_TOKEN }}
      vercel-org-id: ${{ secrets.VERCEL_ORG_ID }}
      vercel-project-id: ${{ secrets.VERCEL_PROJECT_ID }}
```

**Environment Variables:**

# .env.local (development)

NEXT_PUBLIC_ALEO_RPC=https://api.explorer.aleo.org/v1
NEXT_PUBLIC_CONTRACT_ADDRESS=credit_score.aleo
NEXT_PUBLIC_CHAIN_ID=mainnet
NEXT_PUBLIC_WALLET_CONNECT_PROJECT_ID=your_project_id

# Production (Vercel dashboard)

NEXT_PUBLIC_ALEO_RPC=https://api.aleo.org/v1
NEXT_PUBLIC_SENTRY_DSN=https://sentry.io/...
NEXT_PUBLIC_ANALYTICS_ID=G-XXXXXXXXXX

## Monitoring & Observability

**Tools:**

- **Sentry**: Error tracking and performance monitoring
- **Vercel Analytics**: Web Vitals and user behavior
- **PostHog**: Product analytics and feature flags
- **Custom Logging**: Structured logs via Winston

**Sentry Integration:**

```
// lib/sentry.ts
import * as Sentry from '@sentry/nextjs';

Sentry.init({
dsn: process.env.NEXT_PUBLIC_SENTRY_DSN,
environment: process.env.NODE_ENV,
tracesSampleRate: 0.1, // 10% of transactions
beforeSend(event) {
// Strip sensitive data
if (event.request) {
delete event.request.cookies;
delete event.request.headers;
}
return event;
},
});
```

## Performance Monitoring

**Metrics to Track:**

- **Page Load Time**: Target <2s
- **API Response Time**: Target <500ms
- **SDK Operations**: Fetch metrics, generate proof, submit TX
- **Error Rate**: Target <0.1%
- **User Retention**: 1-day, 7-day, 30-day

**Custom Analytics Events:**

```
// lib/analytics.ts
export function trackCreditScoring(event: string, properties: object) {
if (typeof window !== 'undefined') {
window.gtag?.('event', event, {
category: 'Credit Scoring',
```

...properties,
});

```
posthog.capture(event, properties);
```

}
}

// Usage:
trackCreditScoring('score_generated', {
score: 725,
riskLevel: 'medium',
duration: 8500, // ms
});

---

# PART 10: GO-TO-MARKET STRATEGY

## Phase 1: Launch (Weeks 1-4)

**Goals:**

- Deploy to production ✅
- Get first 100 users ✅
- Generate 100+ credit scores ✅
- Zero critical bugs ✅

**Tactics:**

1. **Product Hunt Launch**: Create stunning launch page, gather upvotes
2. **Twitter Announcement**: Thread explaining ProofScore, tag @Aleo
3. **Reddit Posts**: r/CryptoCurrency, r/DeFi, r/AleoOfficial
4. **Discord Outreach**: Aleo Discord, DeFi communities
5. **Demo Video**: 2-minute walkthrough on YouTube

## Phase 2: Growth (Months 2-3)

**Goals:**

- 1,000+ active users
- Partner with 2-3 lending protocols
- $500K+ in undercollateralized loans issued
- Featured on Aleo official channels

**Tactics:**

1. **Hackathon Sponsorship**: Sponsor ETHGlobal, sponsor Aleo Hackathon
2. **Content Marketing**: Blog posts on Medium, technical deep-dives
3. **Influencer Partnerships**: Crypto Twitter influencers, YouTube
4. **Referral Program**: Refer 5 friends → Earn ALEO credits
5. **Protocol Integrations**: Partner with Aave, Compound, Maker

### Phase 3: Scale (Months 4-6)

**Goals:**

- 10,000+ users
- Multi-chain expansion (Ethereum, Polygon, Avalanche)
- Raise seed round ($2-3M)
- Mobile app (iOS + Android)

**Tactics:**

1. **Institutional Partnerships**: Banks, fintechs, DeFi protocols
2. **Token Launch**: ProofScore governance token (optional)
3. **Mobile App**: React Native app for iOS/Android
4. **Cross-Chain Bridge**: Port credits to other blockchains
5. **Media Coverage**: Forbes, CoinDesk, The Block

### Competitive Moats

**Why ProofScore wins long-term:**

1. **Network Effects**: More users = better credit models = more accurate scores
2. **Aleo First-Mover**: Only privacy-first credit scoring on Aleo
3. **Technical Excellence**: Best UX in Web3 credit space
4. **Community Trust**: Open-source SDK, transparent algorithms
5. **Protocol Integrations**: Deep partnerships with lending protocols

---

## PART 11: IMPLEMENTATION ROADMAP

### Week 1-2: Foundation

**Day 1-3: Project Setup**

- [ ] Create Next.js 15 project with TypeScript
- [ ] Install all dependencies (wagmi, RainbowKit, Framer Motion, etc.)
- [ ] Setup Tailwind CSS with custom design system
- [ ] Configure ESLint, Prettier, TypeScript (strict mode)
- [ ] Setup Git repo, commit conventions, PR templates

**Day 4-7: Design System**

- [ ] Implement color palette as CSS variables
- [ ] Create typography system (Inter + JetBrains Mono)
- [ ] Build base UI components (Button, Card, Input, etc.)
- [ ] Setup shadcn/ui components
- [ ] Test responsive breakpoints

**Day 8-10: SDK Core**

- [ ] Implement CreditScoreSDK class structure
- [ ] Build DataAggregator with RPC mocks
- [ ] Implement ScoringEngine with unit tests (100% coverage)
- [ ] Create type definitions for all SDK interfaces

**Day 11-14: Navigation & Landing**

- [ ] Build Navbar with WalletButton
- [ ] Create Hero section with animated gradient text
- [ ] Build Features grid (3 columns)
- [ ] Implement How It Works section
- [ ] Add Footer with social links

## Week 3-4: Core Features

**Day 15-17: Dashboard**

- [ ] Build Dashboard layout
- [ ] Implement MetricsGrid (2x2 responsive)
- [ ] Create MetricCard with animated counters
- [ ] Build ScoreDisplay with animated SVG ring
- [ ] Add ActionButtons (Generate, View, Share)

**Day 18-21: Scoring Flow**

- [ ] Build ScoringModal with 4-step flow
- [ ] Implement FetchingStep (metrics retrieval)
- [ ] Create CalculatingStep (score computation)
- [ ] Build ProvingStep (ZK proof generation)
- [ ] Implement SubmittingStep (blockchain submission)

**Day 22-24: Proof Generation**

- [ ] Implement ProofGenerator class
- [ ] Integrate snarkjs for Groth16 proofs
- [ ] Create witness generation logic
- [ ] Add local proof verification
- [ ] Test with real Aleo addresses

**Day 25-28: Blockchain Integration**

- [ ] Write Leo smart contract (credit_score.aleo)
- [ ] Deploy to Aleo testnet
- [ ] Implement BlockchainAdapter
- [ ] Test transaction submission
- [ ] Add transaction polling logic

## Week 5-6: Polish & Testing

**Day 29-31: Animations**

- [ ] Add Framer Motion to all components
- [ ] Implement staggered animations for lists
- [ ] Create smooth page transitions
- [ ] Add loading states with skeletons
- [ ] Test 60fps performance

**Day 32-35: Mobile Optimization**

- [ ] Test on real iPhone (375px width)
- [ ] Test on real Android device
- [ ] Fix responsive issues
- [ ] Ensure touch targets >44px
- [ ] Test landscape orientation

**Day 36-38: Accessibility**

- [ ] Add ARIA labels to all interactive elements
- [ ] Implement keyboard navigation
- [ ] Test with screen reader (NVDA/VoiceOver)
- [ ] Ensure color contrast >4.5:1
- [ ] Add focus rings to all focusable elements

**Day 39-42: Testing**

- [ ] Write unit tests for SDK (90%+ coverage)
- [ ] Write component tests (70%+ coverage)
- [ ] Write E2E tests (critical flows)
- [ ] Fix all bugs found in testing
- [ ] Run Lighthouse audit (target >90 all)

## Week 7: Deployment

**Day 43-45: Production Prep**

- [ ] Setup Vercel project
- [ ] Configure environment variables
- [ ] Setup custom domain
- [ ] Configure Sentry error tracking
- [ ] Add analytics (Vercel Analytics + PostHog)

**Day 46-47: Launch**

- [ ] Deploy to production
- [ ] Smoke test on production
- [ ] Create demo video
- [ ] Write launch blog post
- [ ] Prepare social media assets

**Day 48-49: Marketing**

- [ ] Post on Product Hunt
- [ ] Tweet launch thread
- [ ] Post in Reddit communities
- [ ] Share in Discord servers
- [ ] Email Aleo team

---

# PART 12: SUCCESS METRICS

## Technical KPIs

**Performance:**

- Page load time: <2 seconds (P95)
- Proof generation time: <3 seconds (P95)
- Transaction submission: <2 seconds (P95)
- Lighthouse score: >90 (all categories)

**Reliability:**

- Uptime: 99.9%+
- Error rate: <0.1%
- Failed transactions: <1%

**Engagement:**

- User retention (7-day): >40%
- Avg session duration: >3 minutes
- Bounce rate: <30%

## Business KPIs

**Adoption:**

- Week 1: 100 users, 100 credit scores
- Month 1: 1,000 users, 1,000+ scores
- Month 3: 10,000 users, 10,000+ scores
- Month 6: 50,000+ users

**Revenue (Future):**

- Protocol integrations: $10K/month per partner
- Premium features: $5/user/month
- API access: $100/month for developers

## Community KPIs

**Social Media:**

- Twitter followers: 1,000+ (Month 1), 10,000+ (Month 6)
- Discord members: 500+ (Month 1), 5,000+ (Month 6)
- GitHub stars: 100+ (Month 1), 1,000+ (Month 6)

**Content:**

- Blog posts: 2 per week
- YouTube videos: 1 per month (tutorials, demos)
- Medium articles: 1 deep-dive per month

---

# REFERENCES


[1] DeFi Pulse. (2026). DeFi Total Value Locked. https://defipulse.com

[2] Alchemy. (2025). Web3 Credit Scoring Tools. https://www.alchemy.com/dapps/best/web3-credit-scoring-tools

[3] Messari. (2025). State of Aleo Q2 2025. https://messari.io/report/state-of-aleo-q2-2025

[4] Calibraint. (2025). Zero Knowledge Proof AI in 2026. https://www.calibraint.com/blog/zero-knowledge-proof-ai-2026

[5] University Mitosis. (2025). ZK Reputation Systems. https://university.mitosis.org/zk-reputation-systems

[6] Oreate AI. (2026). Aleo Mainnet Launch Timeline. https://www.oreateai.com/blog/aleo-mainnet-launch

[7] Equilibrium. (2023). Aleo Deep Dive. https://equilibrium.co/writing/aleo-deep-dive

[8] Deutsche Bank. (2025). Zero-Knowledge Proofs in Blockchain Finance. Corporate Publications

[9] Binance Square. (2025). DeFi Lending in 2026. https://www.binance.com/en/square/post/34503022447898

[10] HES FinTech. (2026). Best Credit Decisioning Software. https://hesfintech.com/blog/best-credit-decisioning-software/