Credit & Funding Software

Watch this video: This is what we will base it off of, except make it 10x better, cleaner, and simpler while keeping the same functions:
https://drive.google.com/file/d/1uahKxMKk9SItifhTiDzwZeW-ZQ4mdJd3/view?usp=drive_link

I have a claude account so just login with my email. For the cursor, just buy it with my card or yours and I'll send the money to you. Anything else just let me know and I'll pay for it. We will get the UI Done at first

We are building a full multi tenant credit analysis and lending platform that allows financial coaches, credit specialists, and lending partners to guide their clients through the entire funding journey in one place. Today, this process is spread across multiple tools that do not communicate with each other: credit report portals, spreadsheets, manual underwriting notes, and external lending links. This creates inefficiency, lost revenue, poor user experience, and very limited scalability.

This platform centralizes every step of the process, from credit intake to funding, into a seamless system designed for both internal use and white label licensing.

Why the Platform Matters

1. **Automates credit intake and analysis**
   Instead of manually reviewing PDF credit reports, the system pulls or parses reports automatically, extracts all key metrics, and generates a readiness evaluation. This gives credit professionals instant insight into whether a client is fundable or needs improvement.

2. **Eliminates guesswork in underwriting**
   The automated readiness engine breaks down utilization, inquiries, score thresholds, public records, and trade lines into a clear pass or fail checklist. This saves hours of manual underwriting and ensures consistency across all clients.

3. **Creates a centralized funding experience**
   Fundable clients enter a Hotwire style lending marketplace where

they see clear funding options without revealing the lender names. This protects proprietary relationships while enabling fast loan selection and tracking.

4. **Drives revenue through automation**
   The system calculates platform fees, lender fees, and partner revenue shares automatically. Every funded loan is tracked end to end with full transparency for admins and partners.

5. **Provides a controlled credit improvement workflow**
   When clients need credit repair, the system routes them to the repair team instantly and logs referral data. No more lost leads, missed opportunities, or manual handoffs.

6. **Gives clients their own portal**
   Clients see their credit metrics, readiness status, recommended actions, event schedule, and basic training content. This boosts engagement and positions the platform as a long term resource, not a one time transaction.

7. **Integrates training, events, and course promotion**
   Light training content helps clients understand next steps, while strategically placed CTAs move users toward premium programs, webinars, or paid events.

8. **Supports scalable white label expansion**
   Partners can run their own branded version of the platform with their own funnels, dashboards, and fee structures. This enables exponential growth without additional operational burden.

9. **Makes the business defensible and ownable**
   Instead of relying on third party tools or someone else's system, this platform becomes proprietary intellectual property that the client fully owns and controls.

Overall Impact

This platform replaces multiple disconnected workflows with one unified system that:

- **Saves time**
- **Increases funding outcomes**
- **Generates recurring revenue**
- **Supports partners**
- **Enhances client education**
- **Improves credit repair conversion**
- **Automates financial decision-making**
- **Positions the client as a leader in the funding and credit industry**

From a development perspective, we are building infrastructure that can scale into a full SaaS product with hundreds of organizations and thousands of clients — all while maintaining clarity, predictability, and strong data governance.

1. Project Overview

1.1 Goal

Build a multi tenant, SaaS style platform for:

- Pulling and analyzing client credit reports
- Routing low score clients into a credit repair path

- Routing qualified clients into a lending marketplace with Hotwire style opaque offers
- Handling admin fees and lender fees
- Supporting white label partners
- Showing a simple training section that pushes users toward the main course
- Showing free and paid events and webinars
- Integrating with one or more credit data vendors plus a fallback PDF upload + AI parsing workflow

All code will live in a monorepo and be built using **Cursor**, with **Claude + OpenAI** as coding and AI processing assistants.

---

2. User Types and Tenancy

2.1 Roles

- SUPER_ADMIN
    - Internal owner of the whole system
    - Can see all organizations, partners, and data
- ORG_ADMIN
    - Client owner (Herman's org) or a partner owner
    - Manages users, clients, referrals, funding, events, training content within their org scope
- AGENT
    - Works client files inside an org
    - Can pull credit reports, run analysis, manage funding events

- CLIENT
    - End customer
    - Logs into their own portal to see:
        - Credit summary
        - Readiness status
        - Paydown suggestions
        - Funding progress
        - Events and basic training content
- PARTNER_ADMIN (White label)
    - Same as ORG_ADMIN but for a partner organization using white label branding

2.2 Tenancy Model

Multi tenant from day one.

- Every org (Herman, plus future partners) has:
    - Unique `organization` record
    - Its own branding
    - Its own users and clients
    - Its own settings (fee splits, thresholds, etc)
- All data is scoped by `organization_id` except global configuration and SUPER_ADMIN views.

---

3. Tech Stack

3.1 High Level

- Monorepo managed by **pnpm**
- Frontend: **Next.js** (latest) with:
    - App Router
    - React
    - Tailwind CSS
    - shadcn/ui components
- Backend: **Node + NestJS** (or Express if devs strongly prefer)
    - Separate app: `apps/api`
    - REST API (JSON)
    - Scheduled jobs for reconciliations
- Database: **PostgreSQL**
    - Access via **Prisma** in a shared package `packages/db`
- Background jobs and queues: **BullMQ + Redis**
    - For long-running tasks such as pulling credit reports, parsing PDFs, polling credit vendors
- Deployment: VPS (DigitalOcean, Hetzner, Linode, etc)
    - API app + Redis + Postgres (managed or self hosted)
    - Frontend can be served from Node or deployed separately
- Auth:
    - JWT based tokens for API
    - NextAuth or custom auth for Next.js, backed by the same users table

- AI:
    - **Claude** and **OpenAI** used for:
        - Parsing credit report blobs into structured JSON
        - Running analysis and generating recommendations
        - Possibly generating dispute letter content in future phases
- Dev environment: **Cursor** IDE with Claude and OpenAI agents connected to repo.

---

4. Branding and UI

- Primary color: `#8faa76`
- Secondary: White
- Text: Black
- Accent red for warnings and failing checks

Design style:

- Clean financial SaaS
- Card-based dashboard layout
- Left sidebar navigation
- Large typography for key metrics
- Icons via Lucide or Heroicons

---

5. Functional Modules

I will describe each module with:

- Purpose
- Core flows
- API endpoints
- Frontend pages
- Notes for AI integration

5.1 Intake and Soft Pull System (Phase 1)

5.1.1 Purpose

Allow org users (agents/admins) to:

- Add a new client
- Initiate a soft pull using a credit vendor API if available
- If no API, upload PDF and let AI parse it
- Store results, then route to Credit Improvement or Lending Marketplace based on score threshold

5.1.2 Flows

1. **Add Client + Intake**

- Admin or Agent opens “New Client” page in dashboard:
    - Fields:
        - First name

        - Last name
        - Email
        - Phone
        - Address
        - Organization (auto scoped)
    - Save creates `client` record.

2. **Authorize Soft Pull**

Two modes (configurable per org):

- API mode:
    - System presents consent language.
    - Fields needed by vendor:
        - SSN
        - DOB
        - Or vendor specific tokens
    - On submit:
        - Create `credit_reports` record with status PENDING
        - Push a job to queue: `CreditPullJob`
- PDF mode:
    - User uploads a PDF credit report file (from IdentityIQ or similar service)
    - Job queued for `CreditParseJob` with file reference

3. **Credit Pull or Parse Job**

- `CreditPullJob`:
    - Use `CreditVendorAdapter` (interface) to call configured vendor (iSoftPull, etc)
    - Save raw JSON response to `credit_reports.raw_data`
    - Status to COMPLETED
    - Enqueue `CreditAnalysisJob`
- `CreditParseJob`:
    - Use an OCR or PDF text extraction library
    - Send extracted text to AI (Claude or OpenAI) with a strict prompt to produce normalized JSON
    - Save normalized JSON into `credit_reports.parsed_data`
    - Status to COMPLETED
    - Enqueue `CreditAnalysisJob`

4. **CreditAnalysisJob**

- Reads `credit_reports.parsed_data`
- Computes:
    - Score (per bureau and summary)
    - Utilization (global and per revolving account)
    - Number of open accounts
    - Inquiries per timeframe
    - Public records and derogatory accounts
- Applies org configuration rules:

    - Minimum score threshold (example 650 or 700)
    - Utilization thresholds
    - Minimum trade lines
- Creates or updates `credit_status`:
    - `path` = CREDIT_REPAIR or LENDING
    - `readiness_score` (0 to 100)
    - `is_fundable` boolean
- Writes logs to `activity_log`.

5.1.3 Backend API

- `POST /api/clients`
- `POST /api/clients/:id/credit/start`
    - Body includes vendor or mode (API vs PDF)
- `POST /api/clients/:id/credit/upload`
    - File upload endpoint
- `GET /api/clients/:id/credit/latest`
    - Returns summary plus parsed data

5.1.4 Frontend

- `/dashboard/clients/new`
- `/dashboard/clients/:id`
    - Shows intake data and “Run credit report” button

- `/dashboard/clients/:id/credit`
    - Shows basic report view and path (Credit Improvement or Lending Marketplace)

5.2 Credit Improvement Path (Phase 2)

5.2.1 Purpose

For clients below fundable threshold:

- Show a clear screen explaining they need credit improvement
- Route them to Herman's credit repair team
- Track referrals and status

5.2.2 Flow

1. After analysis, if path = CREDIT_REPAIR:

- On client page, show prominent banner:
    - "You are not ready for funding yet"
    - Simple explanation: key reasons (high utilization, derogatories, etc)

2. Referral creation:

- Use webhook or API to send client data to external credit repair CRM
- Create `referrals` record with type = CREDIT_REPAIR

3. Status updates:

- Either manual updates via admin
- Or incoming webhook from Herman's CRM to update referral status

5.2.3 Backend API

- `POST /api/clients/:id/referrals/credit-repair`
- `GET /api/referrals?type=credit-repair&organizationId=...`

5.2.4 Frontend

- `/dashboard/clients/:id/credit-improvement`
- Admin view listing all credit repair referrals and statuses

---

5.3 Lending Marketplace Path (Phase 3)

5.3.1 Purpose

For clients who are fundable:

- Show “Hotwire style” opaque funding options
- Support both personal and business funding modes
- Track admin and lender fee calculations

5.3.2 Flow

1. When path = LENDING:

- Display readiness summary:
    - Score, utilization, key checks with green or red icons

2. User chooses funding type:

- Tabs or buttons:
    - “Personal Funding”

    - “Business Funding”

3. Generate offers:

- Backend endpoint receives:
    - Client id
    - Mode (personal or business)
    - Org settings (fee percentages, vendor config)
- Implementation:
    - Initially, mock or static mapping from credit ranges to sample offers
    - Later, integrate with lending marketplace API if available
- Each offer card includes:
    - Loan amount range
    - Estimated APR
    - Estimated term
    - Estimated monthly payment
    - Label like “Best overall”, “Lowest payment”, “Fastest”
- Lender identities remain hidden from client, but stored internally as `lender_id`.

4. Offer selection:

- Agent, Org admin, or Client selects an offer
- System:
    - Persists selection in `offers` table, marks `is_selected = true`
    - Creates a `transactions` record with:
        - Loan amount

        - Platform fee (10 percent)
        - Lender fee (2 percent)
        - Partner share (if any)
        - Status = ESTIMATED

5. Application:

- When offer is selected, show:
    - “Apply now” button which opens external lender application URL
    - After approval, Agent inputs:
        - Approved amount
        - Status: APPROVED or DECLINED
        - Any notes
- Transactions record updated to `status = CONFIRMED` once funded.

5.3.3 Backend API

- `POST /api/clients/:id/offers/generate`
- `GET /api/clients/:id/offers`
- `POST /api/offers/:id/select`
- `POST /api/offers/:id/record-funding`

5.3.4 Frontend

- `/dashboard/clients/:id/offers`
    - Hotwire style grid of opaque offers

- Selection flow with confirmation modal
- Status indicators in client view and admin dashboard

5.4 Admin Dashboard (Phase 4)

5.4.1 Purpose

Give internal team and partner admins visibility into:

- Clients and their statuses
- Credit repair referrals
- Lending offers and funded loans
- Fees and revenue
- Referrals and payouts

5.4.2 Views

- Dashboard home:
    - Total clients
    - Percentage in each path (repair vs lending)
    - Total funding volume
    - Total platform fees generated
    - Total partner shares
    - Total referral payouts
- Clients list:
    - Search, filter by path, readiness, org

- Transactions:
    - Table with:
        - Client name
        - Org name
        - Loan amount
        - Platform fee
        - Lender fee
        - Partner share
        - Status
- Referrals:
    - Table with:
        - Referrer
        - Referred entity (client or partner)
        - Reward amount (e.g. 400)
        - Status
- Partners:
    - List of partner orgs and their metrics

5.4.3 Frontend route

- `/dashboard/admin` with nested views:
    - `/dashboard/admin/clients`
    - `/dashboard/admin/transactions`

- `/dashboard/admin/referrals`
- `/dashboard/admin/partners`

5.5 White Label Platform (Phase 5)

5.5.1 Purpose

Allow multiple partner organizations to use the platform as if it is theirs.

5.5.2 Features

- Organization model includes:
  - Name
  - Logo URL
  - Primary color (optional override)
  - Fee split configuration
  - Subdomain or custom domain mapping
- Scoped views:
  - When an ORG_ADMIN logs in, all data is filtered by their `organization_id`
  - SUPER_ADMIN can choose org from a selector to “impersonate view”
- Partner funnel:
  - Org specific intake links:
    - Example: `https://app.com/h/partner-slug/intake`

5.6 Referral Program (Phase 6)

5.6.1 Purpose

Track and reward affiliates and partners.

5.6.2 Implementation

- `referrals` table includes:
    - referrer_id
    - referred_client_id or referred_organization_id
    - reward_amount
    - status: PENDING, PAYABLE, PAID
    - trigger_event: e.g. LOAN_FUNDED
- Referral links:
    - Format: `https://app.com/r/{referralCode}`
- Nightly job:
    - Check for funded transactions that satisfy conditions
    - Mark referrals as PAYABLE
- Admin dashboard:
    - Controls to mark as PAID and export payout list.

5.7 Training & Course Promotion (Phase 7)

5.7.1 Purpose

Provide light educational content that pushes users toward main course.

5.7.2 Features

- Training page:
    - A list of short lessons or sections:
        - “How to use this platform”
        - “Funding basics”
        - “Credit health basics”
- Each lesson:
    - Title
    - Description
    - Embedded video link or simple text
- Course promotion:
    - Persistent CTA banner:
        - “Want the full system and playbook? Click here to join the main program.”
    - Links to external course platform or funnel.

5.7.3 Frontend

- `/dashboard/training`
- Simple, non complex LMS like layout.

5.8 Events & Webinars (Phase 8)

5.8.1 Purpose

Show upcoming free and paid events, webinars, VIP days.

5.8.2 Features

- `events` table:
    - title
    - description
    - date_time
    - type: FREE or PAID
    - registration_url
    - organization_id
- Events module:
    - Admin can create events
    - Clients and agents see:
        - Next few events on dashboard
        - Full list on dedicated page

5.8.3 Frontend

- `/dashboard/events`
- Dashboard widget showing next events.

5.9 IdentityIQ and Alternative Vendors (Phase 9)

5.9.1 Approach

- IdentityIQ may not provide official API. Treat it as brand integration but not core credit data provider.

- Design `CreditVendorAdapter` interface:

```
interface CreditVendorAdapter {
  pullReport(input: VendorInput): Promise<VendorReportRaw>;
}
```

- Implementations:
    - `ISoftPullAdapter`
    - `SoftPullSolutionsAdapter`
    - `EquifaxAdapter` (future)
- Fallback:
    - PDF upload plus AI parsing as described earlier.

---

6. Data Model (Prisma Outline)

High level tables:

- `User`
- `Organization`
- `Client`
- `CreditReport`
- `CreditStatus`
- `Lender`

- `Offer`
- `Transaction`
- `Referral`
- `Event`
- `ActivityLog`

You can ask Cursor/Claude to generate a full Prisma schema starting from this outline.

---

7. AI Integration Details

7.1 Report Parsing

Use Claude or OpenAI with a prompt like:

- Input: Plaintext credit report (scraped from PDF or vendor)
- Output: Strict JSON with fields like `accounts`, `inquiries`, `scores`, `publicRecords`.

Important:

- Enforce schema in prompt
- Validate JSON server side

7.2 Analysis Summaries

Prompt AI to:

- Explain why client is or is not ready
- Produce bullet point recommendations

- Use neutral tone, no legal advice.

Frontend UI Build Plan

(start here before backend)

1. Core UI Principles

Goals for the look and feel

- Clean financial SaaS, not “template credit repair”
- Uses your brand:
    - Primary: `#8faa76`
    - Backgrounds: white and very light gray
    - Text: black and muted dark gray
- Clear hierarchy:
    - Key metrics big at the top
    - Tables and details below
- Visual language:
    - Green = ready or positive
    - Red = needs attention
    - Yellow or amber = “in progress”

How we differentiate from the competitor UI

- Use a **left sidebar** with icons and labels instead of top-heavy nav

- Use **cards with big headings** and subtle shadows instead of busy panels
- Change naming:
    - “Readiness checklist” instead of “Underwriting” as the main label
    - “Funding options” instead of “Funding audit”
    - “Improvement plan” instead of “Work area”
- Use a slightly different layout:
    - Summary at top, tabs below, then detail sections
- Use different iconography and typography

Functions are similar. Visuals and wording are clearly your own.

---

2. Screen Inventory

Here is the complete list of screens the UI team should create before wiring the backend:

1. Auth
    - Login
    - Forgot password
2. Shell layout
    - Main app layout with sidebar, top bar
3. Admin / Org dashboard
    - Global metrics, charts, quick stats
4. Client list
    - Table of clients with search, filters

5. New client intake
    - Flow to add a client and start credit process
6. Client detail
    - Overview tab
    - Credit report tab
    - Readiness checklist tab
    - Improvement plan tab
    - Funding options tab
7. Opaque funding offers
    - Hotwire style cards for lending
8. Credit improvement view
    - Screen that explains issues and routes to Herman's team
9. Training page
    - Lightweight content + CTAs to main course
10. Events page
    - List of free and paid events / webinars
11. Referral and partner pages
    - Referral link and summary
    - Partner overview (for later integration)
12. Client portal
    - Slimmed down dashboard for end user
    - My credit status

    - My funding status
    - Events and training tiles

You can implement these as static pages first with mocked data.

---

3. Layout and Navigation

3.1 App shell

Create a **main layout component** that all authenticated pages use.

Sidebar items (for admin / org users)

- Dashboard
- Clients
- Funding
- Referrals
- Training
- Events
- Settings

Each item uses an icon and label. On mobile, sidebar collapses.

Top bar

- Organization name and logo
- User avatar menu (profile, sign out)

---

4. Detailed Screen Specs

4.1 Dashboard (Admin / Org)

Purpose
High level overview of what is happening.

Layout

- Top row: four cards
    - Total clients
    - Fundable clients
    - Total funded amount
    - Platform fees generated
- Second row:
    - Left: small chart (bar or line) “Funding volume over time”
    - Right: list “Recent activity” (client X moved to fundable, client Y sent to improvement, etc)
- Third row:
    - Small widget for “Upcoming events” (next 2)
    - Small widget “Training highlight” that links to one featured lesson

All of this can be static JSON for now.

4.2 Clients List

Route: `/dashboard/clients`

Layout

- Header row with:
    - Page title “Clients”

    - Button: “+ New Client”

- Table columns:
    - Client name
    - Email
    - Phone
    - Readiness status (Not ready, In progress, Ready)
    - Path (Credit improvement or Funding)
    - Last updated date
    - Actions (View button)
- Filters:
    - Dropdown filter for Path
    - Search input (by name or email)

4.3 New Client Intake

Route: `/dashboard/clients/new`

Steps

1. Basic info
    - First name
    - Last name
    - Email
    - Phone
    - Address fields

2. Credit pull initiation (UI only for now)
    - Radio buttons:
        - “Pull via integrated credit provider”
        - “Upload credit report PDF”
    - Show consent text
    - Placeholder file upload control if “upload” is selected
3. Summary and “Create client” button

After submit, redirect to client detail page with mock status.

4.4 Client Detail: Overall Layout

Route: `/dashboard/clients/[id]`

Use a content layout with:

- Top card: profile summary
    1. Name, email, phone
    2. Tag: Not ready / Ready for funding
    3. Key metrics:
        - Latest score
        - Utilization
        - Open accounts
- Below that, **tabs**:
    1. Overview
    2. Credit report

3. Readiness checklist
4. Improvement plan
5. Funding options

Each tab just reads static JSON at first.

4.5 Client Detail: Overview Tab

Sections

- “Current status” card
    - Large label: “Ready for funding” or “Needs improvement”
    - Short bullet list:
        - Example: “Score above threshold” / “Utilization too high on 2 cards”
- “Path” card
    - “Current path: Credit improvement” or “Current path: Lending marketplace”
- “Quick actions” buttons
    - “Send to credit improvement team”
    - “View funding options”

This gives a simple snapshot for the agent.

4.6 Credit Report Tab

This is the “read the report” area that mirrors some of what they showed in the call but with your own design.

Layout

- Left side: summary column
    - Scores by bureau in cards (Experian, Equifax, TransUnion)
    - Total accounts
    - Total inquiries
    - Utilization
- Right side: detail area with sub-tabs or sections:
    - Personal info
    - Inquiries
    - Accounts
    - Public records

Each section is a table:

- Inquiries:
    - Date
    - Bureau
    - Creditor
    - Type
- Accounts:
    - Account name
    - Type (revolving, installment, mortgage, etc)
    - Balance
    - Limit
    - Utilization

    - Status (positive, negative, closed, derogatory)

Use color-coded chips for status.

4.7 Readiness Checklist Tab

This is your version of their underwriting red/green screen.

Design

- Big heading: “Readiness checklist”
- Each rule displayed as a row:
    - “Minimum score met”
    - “Total utilization under target”
    - “Enough open trade lines”
    - “No recent late payments”
    - “No major public records”
- Each row has:
    - Label
    - Description or tooltip
    - Status badge:
        - Green “Pass”
        - Red “Needs attention”
- At top, a score ring:
    - “Readiness score: 78 / 100”

This should feel visually different from the other software but functionally parallel.

4.8 Improvement Plan Tab

This corresponds to their “work area”.

Sections

- Summary card:
    - “Key items preventing funding” list
    - Example:
        - High utilization on 2 cards
        - 1 recent late payment
        - 1 collection account
- “Utilization issues” card:
    - Table of accounts that are over threshold
    - Each row has a “View paydown plan” button
- Paydown modal:
    - When clicked, open a modal overlay
    - Shows:
        - Current balance
        - Target balance or utilization
        - Suggested payment amounts:
            - One time
            - Monthly over X months

Right now this can be static math with placeholders, no live calculation.

4.9 Funding Options Tab

This is the Hotwire style part.

Layout

- Top toggle:
    - “Personal”
    - “Business”
- Below, a grid of cards (3 to 5 offers):

Each card shows:

- Title: “Option A” / “Flexible plan” etc
- Amount range: “Up to $50,000”
- Estimated APR: “Starting at 7 percent”
- Term: “36 to 60 months”
- Estimated payment: “Around $XXX / month”
- Tags:
    - “Best overall”
    - “Lowest payment”
    - “Fast decision”
- Button: “Select this option”

At bottom of page:

- Text note:

    - “Exact lenders and terms are shown after selection and qualification.”

This preserves your opaque Hotwire style.

4.10 Credit Improvement Route Screen

Route: `/dashboard/clients/[id]/improvement`

Simple page

- Hero block:
    - “You are not ready for funding yet”
    - Short explanation
- Key issues list (carried from readiness checklist)
- Button:
    - “Send to credit improvement team”
    - “Book call with credit specialist”

This will later trigger the referral logic.

4.11 Training Page

Route: `/dashboard/training`

Sections

- Intro text:
    - Short description about learning how to use the platform and prepare for funding
- Cards for a few lessons (not a full LMS):

    - “Understanding your readiness score”
    - “How fundable clients get more approvals”
    - Each card has a “Watch lesson” button (later embed video links)
- Prominent promotional block:
    - Highlight your main course:
        - “Want the full playbook and deeper training? Join the main program here.”
    - Button linking to course funnel

4.12 Events Page

Route: `/dashboard/events`

Layout

- Title: “Events and trainings”
- Tabs or filters:
    - All
    - Free
    - Paid
- List of event cards:
    - Title
    - Date and time
    - Type (Free / Paid)
    - Short description
    - Button:

        - “Register” or “Learn more”

Also show a small events widget on Dashboard and Client portal.

4.13 Client Portal

Route: `/portal` or `/app/client`

This is what the end user sees.

Differences from agent view:

- Simpler navigation
- No admin or partner sections
- Focused on:
    - My credit snapshot
    - My readiness status
    - My improvement action items
    - My funding status
    - Training tile
    - Events tile

Layout

- Top card:
    - Greeting, readiness status
- Second row:
    - My score and utilization

    - My next step (“Pay down Card X by $Y” or “Schedule a call”)
- Third row:
    - Training highlight
    - Upcoming events

---

5. Reusable Components

Have devs build these as shared primitives early:

- `AppLayout` (with sidebar and topbar)
- `Card` variants (simple, metric, table wrapper)
- `StatusBadge` (Pass, Fail, Attention)
- `Tag` chips (Free, Paid, Personal, Business)
- `DataTable` using shadcn table
- `StatSummary` for showing big metrics
- `TabbedContent` for client detail
- `Modal` component for paydown calculator and confirm dialogs
- `Form` components (with form validation styling)

---

6. Implementation Order for UI

For your team in Cursor:

1. Create the base Next.js app with Tailwind and shadcn

2. Build `AppLayout` and sidebar
3. Build Dashboard UI (static)
4. Build Clients list and New client intake (static)
5. Build Client detail with all tabs (static data)
6. Build Funding options tab with opaque cards
7. Build Training and Events pages
8. Build Client portal UI
9. Polish styling, responsive states, loading skeletons, empty states

Only after UI is solid, start wiring up backend APIs into these screens.

Backend

1. Backend architecture

Apps

- `apps/api`
    - NestJS app
    - REST JSON API
    - Uses Prisma for DB
    - Uses BullMQ for queues

Shared

- `packages/db`
    - Prisma schema and generated client
- `packages/core` (optional)
    - Shared types
    - Utility functions
    - Vendor adapter interfaces

---

2. Data models (Prisma outline)

You will refine this in code, but this is the structure.

```
model Organization {
  id           String   @id @default(cuid())
  name         String
  slug         String   @unique
  logoUrl      String?
  primaryColor String?  // override if needed
  feeSplitPct  Float?   // partner share, if white label

  users        User[]
  clients      Client[]
  events       Event[]
  referrals    Referral[]
  transactions Transaction[]
  createdAt    DateTime @default(now())
  updatedAt    DateTime @updatedAt
}

model User {
  id             String   @id @default(cuid())
  organizationId String
```

```
  organization   Organization @relation(fields: [organizationId],
references: [id])
  email          String   @unique
  passwordHash   String
  role           UserRole
  firstName      String?
  lastName       String?

  referrals      Referral[]      @relation("Referrer")
  createdAt      DateTime @default(now())
  updatedAt      DateTime @updatedAt
}

enum UserRole {
  SUPER_ADMIN
  ORG_ADMIN
  AGENT
  CLIENT
}

model Client {
  id             String   @id @default(cuid())
  organizationId String
  organization   Organization @relation(fields: [organizationId],
references: [id])
  firstName      String
  lastName       String
  email          String
  phone          String?
  addressLine1   String?
  addressLine2   String?
  city           String?
  state          String?
  postalCode     String?
  country        String? @default("US")

  creditReports  CreditReport[]
  creditStatus   CreditStatus?
```

```
  offers         Offer[]
  transactions   Transaction[]
  referrals      Referral[]      @relation("ReferredClient")
  createdAt      DateTime @default(now())
  updatedAt      DateTime @updatedAt
}

model CreditReport {
  id                String   @id @default(cuid())
  clientId          String
  client            Client   @relation(fields: [clientId], references:
[id])
  vendor            String   // "FAKE", "PDF", "ISOTFPULL" later
  status            CreditReportStatus
  rawData           Json?
  parsedData        Json?
  pulledAt          DateTime?
  analysisCompleted Boolean  @default(false)

  createdAt         DateTime @default(now())
  updatedAt         DateTime @updatedAt
}

enum CreditReportStatus {
  PENDING
  COMPLETED
  FAILED
}

model CreditStatus {
  id             String   @id @default(cuid())
  clientId       String   @unique
  client         Client   @relation(fields: [clientId], references:
[id])
  path           CreditPath
  readinessScore Int
  isFundable     Boolean
  notes          String?
```

```
  lastAnalysisAt DateTime @default(now())
}

enum CreditPath {
  CREDIT_REPAIR
  LENDING
}

model Lender {
  id           String   @id @default(cuid())
  name         String
  internalCode String   @unique
  websiteUrl   String?
  isActive     Boolean  @default(true)
  meta         Json?

  offers       Offer[]
}

model Offer {
  id             String   @id @default(cuid())
  clientId       String
  client         Client   @relation(fields: [clientId], references:
[id])
  organizationId String
  organization   Organization @relation(fields: [organizationId],
references: [id])
  lenderId       String?
  lender         Lender?  @relation(fields: [lenderId], references:
[id])
  mode           FundingMode // PERSONAL or BUSINESS

  amountMin      Int
  amountMax      Int
  aprMin         Float
  aprMax         Float
  termMinMonths  Int
```

```
  termMaxMonths  Int
  estMonthlyPay  Float

  label          String? // "Best overall" etc
  isSelected     Boolean @default(false)
  status         OfferStatus @default(GENERATED)

  createdAt      DateTime @default(now())
  updatedAt      DateTime @updatedAt
}

enum FundingMode {
  PERSONAL
  BUSINESS
}

enum OfferStatus {
  GENERATED
  SELECTED
  FUNDED
  DECLINED
}

model Transaction {
  id             String   @id @default(cuid())
  clientId       String
  client         Client   @relation(fields: [clientId], references:
[id])
  organizationId String
  organization   Organization @relation(fields: [organizationId],
references: [id])
  offerId        String?
  offer          Offer?   @relation(fields: [offerId], references:
[id])

  loanAmount     Int
  platformFee    Float    // 10 percent of loanAmount for now
  lenderFee      Float    // 2 percent of loanAmount
```

```
  partnerShare   Float?   // computed from feeSplitPct
  status         TransactionStatus

  createdAt      DateTime @default(now())
  updatedAt      DateTime @updatedAt
}

enum TransactionStatus {
  ESTIMATED
  CONFIRMED
  PAID_OUT
}

model Referral {
  id                 String   @id @default(cuid())
  organizationId     String
  organization       Organization @relation(fields: [organizationId],
references: [id])
  referrerId         String
  referrer           User     @relation("Referrer", fields:
[referrerId], references: [id])
  referredClientId   String?
  referredClient     Client?  @relation("ReferredClient", fields:
[referredClientId], references: [id])
  referredOrgId      String?
  referredOrg        Organization? @relation("ReferredOrg", fields:
[referredOrgId], references: [id])

  type               ReferralType
  rewardAmount       Float
  status             ReferralStatus
  triggerEvent       String? // e.g. "LOAN_FUNDED_OVER_XX"
  createdAt          DateTime @default(now())
  updatedAt          DateTime @updatedAt
}

enum ReferralType {
  CREDIT_REPAIR
```

```
  FUNDING
  WHITE_LABEL
}

enum ReferralStatus {
  PENDING
  PAYABLE
  PAID
}

model Event {
  id             String   @id @default(cuid())
  organizationId String
  organization   Organization @relation(fields: [organizationId],
references: [id])
  title          String
  description    String?
  startTime      DateTime
  type           EventType // FREE or PAID
  registrationUrl String?

  createdAt      DateTime @default(now())
  updatedAt      DateTime @updatedAt
}

enum EventType {
  FREE
  PAID
}

model ActivityLog {
  id             String   @id @default(cuid())
  organizationId String
  organization   Organization @relation(fields: [organizationId],
references: [id])
  entityType     String
  entityId       String
  action         String
```

```
  meta           Json?
  createdAt      DateTime @default(now())
}
```

This is enough to start backend code.

---

3. NestJS modules and responsibilities

Set up modules like this:

- `AuthModule`
- `OrganizationsModule`
- `UsersModule`
- `ClientsModule`
- `CreditModule`
- `OffersModule`
- `TransactionsModule`
- `ReferralsModule`
- `EventsModule`
- `AdminModule` (aggregated stats)
- `JobsModule` (BullMQ queues)
- `VendorsModule` (adapters and placeholder providers)

Each module gets a controller, service, DTOs, and uses Prisma.

---

4. Key endpoints with placeholder behavior

4.1 Auth

Basic JWT for now.

- `POST /auth/login`
    - Request: email, password
    - Response: token, user, organization
- `POST /auth/register` (for initial seed or internal use only)

Middlewares or guards:

- `AuthGuard` to require JWT
- Role based guard: `RolesGuard(SUPER_ADMIN, ORG_ADMIN, AGENT, CLIENT)`

4.2 Organizations and Users

These are mostly CRUD for admin use. Can be built later. Keep it simple:

- `GET /organizations/me`
- `GET /organizations` (SUPER_ADMIN only)
- `GET /users/me`
- `GET /users`
- `POST /users`
- `PATCH /users/:id`

4.3 Clients

`ClientsModule`

- `GET /clients`
    - Query params: organization, path, readiness, search
- `POST /clients`
    - Create client record
- `GET /clients/:id`
    - Returns client info, latest credit status summary
- `DELETE /clients/:id` (later if needed)

4.4 Credit intake, pull, parse, analysis

`CreditModule`

4.4.1 Start credit process

- `POST /clients/:id/credit/start`

Request body idea:

```
{
  "mode": "VENDOR_API", // or "PDF"
  "vendor": "FAKE", // placeholder, later "ISOFTPULL" etc
  "input": {
    "ssn": "XXX-XX-XXXX",
    "dob": "YYYY-MM-DD"
  }
}
```

Behavior now:

- Validate client belongs to current org
- Create `CreditReport` row with status PENDING, vendor "FAKE"
- Enqueue `CreditPullJob` with `creditReportId` and requested vendor

For now, do not call any real external API in this job. Use a placeholder.

4.4.2 Upload credit report PDF

- `POST /clients/:id/credit/upload`
    - Multipart with `file`

Behavior:

- Save file path or storage key in `CreditReport`
- Set vendor = "PDF"
- Status = PENDING
- Enqueue `CreditParseJob`

Again, no real vendor involved. Just stub.

4.4.3 Get latest credit summary

- `GET /clients/:id/credit/latest`

Response:

```
{
  "creditReportId": "xxx",
  "status": "COMPLETED",
  "score": {
    "summary": 705,
    "experian": 700,
    "equifax": 710,
```

```
    "transunion": 705
  },
  "utilization": {
    "overall": 28.5,
    "revolving": 32.1
  },
  "openAccounts": 7,
  "derogatories": 1,
  "path": "LENDING",
  "readinessScore": 82,
  "isFundable": true
}
```

Initially this should be mock data from `parsedData` or stubbed JSON.

4.5 Offers and funding options

`OffersModule`

4.5.1 Generate offers

- `POST /clients/:id/offers/generate`

Request body:

```
{
  "mode": "PERSONAL"
}
```

Behavior now:

- Check client has a completed `CreditReport` and `CreditStatus`
- Use simple internal logic to create 3 to 5 `Offer` rows using static templates based on readiness score and maybe organization config.

For example:

- If readinessScore > 80
    - generate higher amounts and lower APR ranges
- If 60 to 80
    - smaller amounts or wider APR ranges

No external lender API yet, just placeholders.

Response:

```
{
  "offers": [
    {
      "id": "offer1",
      "label": "Best overall",
      "amountMin": 20000,
      "amountMax": 50000,
      "aprMin": 7.0,
      "aprMax": 13.5,
      "termMinMonths": 36,
      "termMaxMonths": 60,
      "estMonthlyPay": 950,
      "mode": "PERSONAL"
    }
  ]
}
```

4.5.2 List offers

- `GET /clients/:id/offers`

Returns all offers for that client.

4.5.3 Select offer

- `POST /offers/:id/select`

Behavior:

- Mark that offer `isSelected = true`, `status = SELECTED`
- Create `Transaction` with:
    - `loanAmount` = average of amountMin and amountMax for now
    - `platformFee` = loanAmount * 0.10
    - `lenderFee` = loanAmount * 0.02
    - `partnerShare` = computed if org has `feeSplitPct`

Return transaction summary.

4.5.4 Record funding result

- `POST /offers/:id/record-funding`

Request body:

```
{
  "loanAmount": 25000,
  "status": "CONFIRMED"
}
```

Behavior:

- Update Offer status to FUNDED or DECLINED
- Update Transaction loanAmount and status
- Emit ActivityLog row

4.6 Transactions

`TransactionsModule`

Most data is created from offer selection.

- `GET /transactions`
    - Query: organization, client, status
- `GET /transactions/:id`

Later add export endpoints.

4.7 Referrals

`ReferralsModule`

4.7.1 Create credit repair referral

- `POST /clients/:id/referrals/credit-repair`

For now, no external CRM call. Just:

- Create `Referral` row with type CREDIT_REPAIR, rewardAmount placeholder (0 or configured), status PENDING
- Emit ActivityLog

Later, you add webhook to Herman's CRM.

4.7.2 List referrals

- `GET /referrals`

4.8 Events

`EventsModule`

- `POST /events`
- `GET /events`
- `GET /events/:id`
- `PATCH /events/:id`

Simple CRUD, scoped by organization.

4.9 Admin stats

`AdminModule`

One endpoint that the dashboard will hit:

- `GET /admin/summary`

Returns:

```
{
  "totalClients": 120,
  "fundableClients": 54,
  "totalFundedAmount": 1350000,
  "platformFees": 135000,
  "partnerShares": 54000
}
```

Right now, can be very simple aggregate Prisma queries.

---

5. Jobs and queues

Use BullMQ.

Create a `JobsModule` with:

- Queue `credit-pull`
- Queue `credit-parse`
- Queue `credit-analysis`
- Queue `referral-check` (for nightly referral payout eligibility later)

5.1 CreditPullJob

Worker does:

- Load CreditReport by id
- If vendor is "FAKE":
    - Sleep a second
    - Write stub `rawData` JSON with fake but plausible credit numbers
    - Set status COMPLETED
    - Enqueue `credit-analysis`

Later this is where you drop in real vendor code.

5.2 CreditParseJob

Worker:

- Load PDF location from CreditReport
- For now, ignore file and just write stub `parsedData` from a fixture.
- Mark status COMPLETED
- Enqueue `credit-analysis`

Later you plug in OCR and AI parsing.

5.3 CreditAnalysisJob

Worker:

- Read `rawData` or `parsedData` from CreditReport
- If stub, use internal function to compute sample numbers
- Create or update `CreditStatus`
- Path is decided by readinessScore and min score threshold config
- Write ActivityLog

---

6. Vendor adapters with placeholders

Create in `VendorsModule` a simple interface:

```
export interface CreditVendorAdapter {
  pullReport(input: any): Promise<any>;
}
```

Create an implementation `FakeCreditVendorAdapter`:

```
@Injectable()
export class FakeCreditVendorAdapter implements CreditVendorAdapter {
  async pullReport(input: any): Promise<any> {
    // TODO replace with real vendor call
    // For now return stub data
    return {
      scores: {
        experian: 705,
        equifax: 710,
        transunion: 700
      },
      utilization: {
        overall: 28.5,
```

```
        revolving: 32.1
      },
      accounts: [],
      inquiries: [],
      publicRecords: []
    };
  }
}
```

Wire this into `CreditService` with dependency injection. Later, when you integrate iSoftPull or another provider, create `ISoftPullAdapter` and swap through a config or feature flag.

---

7. Config and environment

Create a config module to read from `.env`:

- `DATABASE_URL`
- `JWT_SECRET`
- `CREDIT_VENDOR` // "FAKE", later "ISOFTPULL" etc
- `QUEUE_REDIS_URL`

In `CreditService`:

```
constructor(
  @Inject(CREDIT_VENDOR_TOKEN) private vendor: CreditVendorAdapter,
  private prisma: PrismaService
) {}
```

And bind the token based on env.