Part A: User Stories and On-Chain Requirements

1) Users

1. **Direct Users:**
    - **Business/Merchant:** Creates a Subscription, defines the payment terms (token, price, frequency and tracks overall revenue.
    - **Subscriber:** Initiates the subscription created by the merchant, delegates the token account to the service for future trustless deductions.
2. **Indirect Users:**
    - **Automation Operator:** This external service monitors all active subscriptions and acts as the time-based trigger for the system. It utilizes the delegated authority to autonomously move tokens from the Subscriber.
3. **Administrators / Moderators:**
    - **Protocol Developer:** Writes and upgrades the core Anchor program logic, manages the PDA seeds, and handles administrative tasks like potentially pausing the service or updating fees.
4. **Stakeholders:**
    - **Venture Capital/Investors:** This group represents the financial stakeholders who recognize the strategic value of enabling decentralized recurring revenue on Solana.

2) User Stories

1. **The Enrollment and Authorization Flow:** Merchant can create a subscription plan with the required parameters for a subscription and Subscriber (who will subscribe) will delegate his token account to program for automated payments.

User Type	Action	Expected Outcome
Merchant	When I create a new service plan with a token, price, and monthly frequency	A new Payment Plan is instantly visible to the public, detailing the subscription terms and the Merchant's designated collection wallet.
Subscriber	When I choose a plan and click "Subscribe" on the dApp	My wallet prompts me with a clear, single request to authorize future token transfers up to the required subscription limit by the service.
Subscriber	After I approve the future token transfers in my wallet	A Subscription Confirmation page appears, showing the plan details, the first payment date, and a notice that my monthly payments are now automated.

Subscriber	When I view my active subscriptions	I see the plan name, the token being used, the total amount of funds I have authorized the service to spend, and the next scheduled payment date.

Potential On-Chain Requirements:

a) **create_subscription Instruction:** Needs to accept and store the price (u64), frequency_days (u16), and token_mint (Pubkey), (other things maybe added).
b) **Account Initialization:** Must initialize the SubscriptionPlan PDA, keyed by the Merchant's authority and a unique ID (e.g., ['plan', merchant_key, plan_id]), and enforce that only the Merchant is the signer.
c) **Token Account Check:** Must verify that the provided Merchant Revenue Token Account is associated with the correct token_mint and is owned by the Merchant.
d) **subscribe Instruction:** Needs to initialize the UserSubscription PDA, keyed by the Subscriber's wallet and the plan.
e) **Initial State Setup:** The subscribe instruction must calculate and store the next_payment_date (current timestamp + frequency_days).
f) **Delegation Check:** Before finalizing the subscription, the client must ensure the SPL Token Program registers the Merchant's Service Authority PDA as a delegate over the Subscriber's token account.
2. **The Automated Recurring Deduction Cycle:** Offchain Automation Operator can trigger the on-chain smart contract to successfully move tokens using the prior delegation, without requiring a live user signature.

User Type	**Action**	**Expected Outcome**
Automation Operator	When the system identifies a user whose next payment date has arrived	A successful, zero-fee transaction occurs in the background, and the payment amount is instantly transferred from the user's wallet to the Merchant's collection wallet.
Merchant	After a scheduled payment is successfully processed by the system	My collection wallet balance increases by the correct amount, and the user's next payment date is automatically advanced by one payment interval (e.g., 30 days).

Potential On-Chain Requirements:

a) **charge_user Instruction:** Needs to require the following accounts: UserSubscription PDA, SubscriptionPlan PDA, Subscriber's Token Account, Merchant's Revenue Token Account, SPL Token Program ID, and the Service Authority PDA.

b) **Time Validation:** The instruction must contain a constraint checking that the current time (Clock::get()) is greater than or equal to the next_payment_date stored on the UserSubscription account.
c) **Access Control:** Must enforce that the Automation Operator's Public Key is a required signer for the instruction, preventing unauthorized calls.
d) **CPI Execution:** Execute a Cross-Program Invocation to the SPL Token Program's transfer instruction, using the Service Authority PDA's seeds to sign the transaction, drawing tokens from the Subscriber's account.
e) **State Update:** If the CPI succeeds, the instruction must update the UserSubscription account by advancing the next_payment_date by the frequency_days defined in the SubscriptionPlan.
f) **Error Handling:** If the CPI fails (e.g., due to insufficient delegation or balance), the instruction should not panic. Instead, it should log the specific error code and update the UserSubscription account's status to Failed and increment a failure_count.

3) On-Chain Requirements

1) **Program Requirements**
    a) **Delegation CPI Implementation:** The program must include the charge_user instruction capable of executing a Cross-Program Invocation (CPI) to the SPL Token Program. This CPI must perform a transfer instruction, signed by the Service's Program Derived Address (PDA), using the delegated authority over the user's token account.
    b) **State Management Logic:** Implement instructions (subscribe, charge_user) that accurately calculate and update the next_payment_date based on the frequency defined in the SubscriptionPlan account.
    c) **Access Control:** Ensure only the Merchant can create and update the SubscriptionPlan and that only the Automation Operator can successfully call the charge_user instruction.
    d) **Error Handling:** Implement checks in charge_user to gracefully fail and log if the Subscriber has revoked delegation or if the token account has insufficient balance.
2) **Account Requirements**
    a) **SubscriptionPlan PDA:** A state account storing the immutable terms of the service (e.g., price: u64, frequency_days: u16, token_mint: Pubkey).
    b) **UserSubscription PDA:** The primary state tracker, keyed by the Subscriber's wallet and the SubscriptionPlan. It must store the Subscriber's token account and the next_payment_date.

c) **Service Authority PDA (The Delegate):** The specific Program Derived Address that is designated to receive the SPL Token approve instruction from the user. This PDA will be the signer for all automated transfers.
d) **Merchant Revenue Token Account:** A designated SPL Token account owned by the Merchant where all successful payments are deposited.

3) **On-Chain Account Schema:**

Account	**Purpose & PDA Seeds**	**Required Data Fields**
SubscriptionPlan	Stores immutable service details. Seeds: ['plan', merchant_key, plan_id_u32]	merchant_key: Pubkey, token_mint: Pubkey, price: u64, frequency_days: u16, revenue_token_account: Pubkey
UserSubscription	Stores the Subscriber's active state. Seeds: ['subscription', subscriber_key, subscription_plan_key]	subscriber_key: Pubkey, subscriber_token_account: Pubkey, next_payment_date: i64, status: SubscriptionStatus (enum), failure_count: u8
Service Authority PDA	The non-wallet account that acts as the SPL delegate signer. Seeds: ['authority']	No stored data; its existence and key are used for signing.

4) **Off-Chain Requirements**

Automation Service: A dedicated, off-chain service responsible for:

- **Scanning:** Periodically querying Solana to identify all UserSubscription accounts where the current time is past the next_payment_date.
- **Transmitting:** Constructing and submitting the charge_user transaction for each due subscription. This service must hold the private key of the authorized Operator wallet.

Part B: Process Appendix

Critical User Selection Rationale

- **Merchant:** Must define the service and set the economic terms (price, token).
- **Subscriber:** Must perform the crucial **SPL Token approve delegation**, which is the core innovation.
- **Automation Operator:** Must execute the charge_user instruction, validating the **"automated"** part of the value prop.
- **Protocol Developer:** Manages the on-chain logic and security.

Requirements Granularity and Refinement

- **User Story Rationale:** User stories were crafted to follow the INVEST model, focusing on the user's action and value rather than technical concepts like CPIs or PDAs.
- **Refinement Area: State Tracking**
  - **Initial Requirement / Problem:** Only tracking next_payment_date.
  - **Final Granular Requirement:** Added status: SubscriptionStatus and failure_count: u8 to UserSubscription for reliable business logic and retry mechanisms.
- **Refinement Area: Failure Logging**
  - **Initial Requirement / Problem:** Vague "system logs a status" upon failure.
  - **Final Granular Requirement:** Must explicitly **log the specific SPL error code** and update the UserSubscription account's status to Failed upon CPI failure.
- **Refinement Area: Account Identity**
  - **Initial Requirement / Problem:** Only conceptual PDAs (e.g., "SubscriptionPlan PDA").
  - **Final Granular Requirement:** Defined explicit Anchor PDA seeds: ['plan', merchant_key, plan_id], ['subscription', subscriber_key, plan_key], ensuring deterministic address lookups.
- **Refinement Area: Automation Security**
  - **Initial Requirement / Problem:** Unclear who can call charge_user.
  - **Final Granular Requirement:** Enforced a strict **Access Control** check in charge_user requiring the fixed **Automation Operator's Pubkey** as the instruction signer.

Part C Refinement Log

Original Story 1 (Subscriber)

- **Before Story:** When I choose a plan and click "Subscribe" on the dApp, My wallet prompts me with a clear, single request to authorize future token transfers up to the required subscription limit by the service.
- **After Story (NEW Story 2A):** When I click "Authorize Payments" on a plan, my wallet asks me to **delegate the maximum amount of tokens** this service can spend from my account.
- **Rationale for Change: Atomicity & Clarity.** The user performs two distinct signed actions. This story is now solely focused on the **Delegation (SPL approve)** action and its outcome (the wallet prompt).

Original Story 2 (Subscriber)

- **Before Story:** After I approve the future token transfers in my wallet, A Subscription Confirmation page appears, showing the plan details, the first payment date, and a notice that my monthly payments are now automated.
- **After Story (NEW Story 2B):** After successfully authorizing the funds, when I click "Finalize Subscription," a **Subscription Confirmation Page** loads, displaying the date of my next automated payment.
- **Rationale for Change: Atomicity & Clarity.** This story is now solely focused on the **Subscription Creation (Program subscribe)** action. The user's successful interaction with the program's Anchor instruction is the only action being tested here.

Original Story 3 (Subscriber)

- **Before Story:** When I view my active subscriptions, I see the plan name, the token being used, the total amount of funds I have authorized the service to spend, and the next scheduled payment date.
- **After Story (Refined Story 2C):** When I view my active subscriptions, I see the plan name, the token being used, the **status of my funds delegation**, and the next scheduled payment date.
- **Rationale for Change: De-Jargon/Clarity.** "Total amount of funds I have authorized" is a technical detail. Reframed the outcome to the necessary business status: the current **delegation status** (Active, Revoked, Expired) is visible.

AI Prompts and Results:

> **Prompt:** For a project with this value proposition
The project is an on-chain, automated subscription payment soution using bockchain. It aows businesses to create and manage recurring payment pans (e.g., monthy subscriptions) entirey within the decentraized ecosystem. Cruciay, it wi utiise the

native SPL Token deegation mechanism, enabing the service's Program Derived Address PDA) to automaticay deduct tokens from a user's waet on a set schedue without continuous user interaction or active signatures. and focusing
on these prioritized user types
Direct Users:
Business/Merchant: Creates a Subscription, defines the payment terms (token, price, frequency and tracks overall revenue.
Subscriber: Initiates the subscription created by the merchant, delegates the token account to the subscription service for future trustless deductions.
Indirect Users:
Automation Operator: This external service monitors all active subscriptions and acts as the time-based trigger for the system. It utilizes the delegated authority to autonomously move tokens from the Subscriber.
Administrators / Moderators:
Protocol Developer: Writes and upgrades the core Anchor program logic, manages the PDA seeds, and handles administrative tasks like potentially pausing the service or updating fees.
Stakeholders:
Venture Capital/Investors: This group represents the financial stakeholders who recognize the strategic value of enabling decentralized recurring revenue on Solana.
help map out the key functions or interactions each user would
need to perform.

> **Output:** This is the crucial step that translates your user roles into concrete smart contract interactions and off-chain service functions. By mapping these actions, you define the core API of your project.
I have mapped the key functions for the three active user types (Merchant, Subscriber, and Operator). I've excluded the Protocol Developer (as their functions are covered by the other three) and the Investors (as they perform no functional interaction with the service).

Key User Functions and Interactions
1. Subscription Service Administrator (The Merchant)
The Merchant's functions focus on creation, configuration, and monitoring—all essential for launching a monetized product.

Function	Type	Description
initialize_service	On-Chain (Anchor Instruction)	Initializes the global service PDA and sets the initial administrative fee/metadata.
create_subscription_pla n	On-Chain (Anchor Instruction)	Defines a specific product tier: sets the SPL Token mint, price, and payment frequency (e.g., 30 days).

update_subscription_plan	On-Chain (Anchor Instruction)	Modifies the price or frequency of an existing plan for future subscribers.
Monitor Revenue	Off-Chain (Analytics)	Observes the balance and transaction history of the Revenue Account to track aggregate income.

2. Subscriber (The Paying Customer)

The Subscriber's functions are centered around initial authorization and managing their delegation status.

Function	Type	Description
SPL Token Approve	On-Chain (SPL Token Instruction)	The Critical Step: Executes the SPL instruction to delegate spending authority (up to a limit or indefinitely) over the user's token account to the service's PDA.
subscribe	On-Chain (Anchor Instruction)	Creates the user's personal UserSubscription account, linking them to a SubscriptionPlan and recording the next_payment_date.
cancel_subscription	On-Chain (Anchor Instruction)	Ends the recurring payment by marking the UserSubscription as inactive.
SPL Token Revoke	On-Chain (SPL Token Instruction)	Revokes the delegation authority from the service's PDA, providing an emergency stop for payments.

3. Automation/Bot Operator (Off-Chain Infrastructure)

The Operator's functions are purely administrative and time-driven, validating the system's "automated" core value.

Function	Type	Description
Identify Due Subscriptions	Off-Chain (Database Query)	Scans the list of active UserSubscription accounts to find those where the current date ≥ next_payment_date.
charge_user	On-Chain (Anchor Instruction)	For each due subscription, executes the instruction. The program uses the PDA as the signer to transfer tokens from the delegated user account to the Merchant's Revenue Account.

Handle Errors	Off-Chain (Service Logic)	Manages failed transactions (e.g., due to insufficient delegated funds or an empty user account) and logs the failure status on-chain.

> **Prompt:** Based on these top 1-2 critical user interactions
The Enrollment and Authorization Flow: Merchant can create a subscription plan with the required parameters for a subscription and Subscriber (who will subscribe) will delegate his token account to program for automated payments.
User
Function
Outcome Validation
Merchant
create_subscription_plan
A new, specific subscription plan (price, token, frequency) exists on-chain.
Subscriber
SPL Token Approve
The Subscription Service PDA is now the authorized delegate over the Subscriber's token account.
Subscriber
subscribe
A UserSubscription account is created, recording the next payment date and confirming the delegation status.
The Automated Recurring Deduction Cycle: Offchain Automation Operator can trigger the on-chain smart contract to successfully move tokens using the prior delegation, without requiring a live user signature.
User
Function
Outcome Validation
Automation Operator
Identify Due Subscriptions (Off-Chain)
A specific UserSubscription is identified as ready for payment based on the stored next_payment_date.
Automation Operator
charge_user (On-Chain)
The PDA signs the SPL Token transfer instruction, tokens move from the Subscriber's delegated account to the Merchant's Revenue Account, and the next_payment_date is successfully updated.
what are the key technical requirements needed to build a
Proof-of-concept?

Output: This is a crucial breakdown. By focusing on the core interactions, we can define a tight set of technical requirements that validate your primary value proposition: automated, trustless payment deduction.

The technical requirements for your Proof-of-Concept (PoC) fall into three critical areas: On-Chain Logic, Solana Accounts, and Off-Chain Infrastructure.

Key Technical Requirements for the Proof-of-Concept

1. On-Chain Program Requirements (Anchor)

These requirements define the core API and the critical mechanism of automated deduction.

- Delegation CPI Implementation: The program must include the charge_user instruction capable of executing a Cross-Program Invocation (CPI) to the SPL Token Program. This CPI must perform a transfer instruction, signed by the Service's Program Derived Address (PDA), using the delegated authority over the user's token account.
- State Management Logic: Implement instructions (subscribe, charge_user) that accurately calculate and update the next_payment_date based on the frequency defined in the SubscriptionPlan account.
- Access Control: Ensure only the Merchant can create and update the SubscriptionPlan and that only the Automation Operator can successfully call the charge_user instruction (e.g., by checking the caller is a specific authority or bot wallet).
- Error Handling: Implement checks in charge_user to gracefully fail and log if the Subscriber has revoked delegation or if the token account has insufficient balance.

2. Solana Account Requirements

The system requires several distinct on-chain data accounts to store state and act as signatories.

- SubscriptionPlan PDA: A state account storing the immutable terms of the service (e.g., price: u64, frequency_days: u16, token_mint: Pubkey).
- UserSubscription PDA: The primary state tracker, keyed by the Subscriber's wallet and the SubscriptionPlan. It must store the Subscriber's token account and the next_payment_date.
- Service Authority PDA (The Delegate): The specific Program Derived Address that is designated to receive the SPL Token approve instruction from the user. This PDA will be the signer for all automated transfers.
- Merchant Revenue Token Account: A designated SPL Token account owned by the Merchant where all successful payments are deposited.

3. Off-Chain Infrastructure Requirements

These requirements ensure the "automated" part of the value proposition is fully realized outside the Solana program.

- Subscriber Client Logic (Critical): The frontend (dApp) must first construct and submit the SPL Token approve instruction (not an Anchor instruction) before calling your program's subscribe function. This step is where the user delegates authority.
- Automation Service (Keeper/Cron Job): A dedicated, off-chain service (e.g., a simple Node.js application or a commercial keeper service) responsible for:

1. Scanning: Periodically querying Solana to identify all UserSubscription accounts where the current time is past the next_payment_date.
2. Transmitting: Constructing and submitting the charge_user transaction for each due subscription. This service must hold the private key of the authorized Operator wallet.

Prompt: Review my core user functions/stories 2) User Stories
The Enrollment and Authorization Flow: Merchant can create a subscription plan with the required parameters for a subscription and Subscriber (who will subscribe) will delegate his token account to program for automated payments.
User Type
Action
Expected Outcome
Merchant
When I create a new service plan with a token, price, and monthly frequency
A new Payment Plan is instantly visible to the public, detailing the subscription terms and the Merchant's designated collection wallet.
Subscriber
When I choose a plan and click "Subscribe" on the dApp
My wallet prompts me with a clear, single request to authorize future token transfers up to the required subscription limit by the service.
Subscriber
After I approve the future token transfers in my wallet
A Subscription Confirmation page appears, showing the plan details, the first payment date, and a notice that my monthly payments are now automated.
Subscriber
When I view my active subscriptions
I see the plan name, the token being used, the total amount of funds I have authorized the service to spend, and the next scheduled payment date.
The Automated Recurring Deduction Cycle: Offchain Automation Operator can trigger the on-chain smart contract to successfully move tokens using the prior delegation, without requiring a live user signature.
User Type
Action
Expected Outcome
Automation Operator
When the system identifies a user whose next payment date has arrived
A successful, zero-fee transaction occurs in the background, and the payment amount is instantly transferred from the user's wallet to the Merchant's collection wallet.
Merchant
After a scheduled payment is successfully processed by the system
My collection wallet balance increases by the correct amount, and the user's next payment date is automatically advanced by one payment interval (e.g., 30 days).
and requirements Program Requirements

Delegation CPI Implementation: The program must include the charge_user instruction capable of executing a Cross-Program Invocation (CPI) to the SPL Token Program. This CPI must perform a transfer instruction, signed by the Service's Program Derived Address (PDA), using the delegated authority over the user's token account.
State Management Logic: Implement instructions (subscribe, charge_user) that accurately calculate and update the next_payment_date based on the frequency defined in the SubscriptionPlan account.
Access Control: Ensure only the Merchant can create and update the SubscriptionPlan and that only the Automation Operator can successfully call the charge_user instruction.
Error Handling: Implement checks in charge_user to gracefully fail and log if the Subscriber has revoked delegation or if the token account has insufficient balance.
Account Requirements
SubscriptionPlan PDA: A state account storing the immutable terms of the service (e.g., price: u64, frequency_days: u16, token_mint: Pubkey).
UserSubscription PDA: The primary state tracker, keyed by the Subscriber's wallet and the SubscriptionPlan. It must store the Subscriber's token account and the next_payment_date.
Service Authority PDA (The Delegate): The specific Program Derived Address that is designated to receive the SPL Token approve instruction from the user. This PDA will be the signer for all automated transfers.
Merchant Revenue Token Account: A designated SPL Token account owned by the Merchant where all successful payments are deposited.
Off-Chain Requirements
Automation Service: A dedicated, off-chain service responsible for:
Scanning: Periodically querying Solana to identify all UserSubscription accounts where the current time is past the next_payment_date.
Transmitting: Constructing and submitting the charge_user transaction for each due subscription. This service must hold the private key of the authorized Operator wallet..
Considering my project's
refined value proposition, do these
stories truly hit the mark? Are the requirements granular enough to map
to specific technical components (e.g., database schemas, API endpoints,
specific blockchain programs)? What's missing or unclear?

Output: This is an excellent set of user stories and requirements. They are highly focused on the core innovation—SPL delegation—and demonstrate a clear understanding of the full stack (on-chain, client, and off-chain automation).
To ensure they are truly ready for development and meet the standard of being granular enough to map to specific technical components, here is a detailed review highlighting what's working well, what's missing, and where clarity is needed.

Review of User Stories and Technical Requirements
1. Strengths (What's Working)

Component	Assessment
Focus on Delegation	Excellent. Both core stories are centered on the success of the SPL approve and the PDA-signed transfer. This directly validates your core value proposition (automated, trustless payment).
Separation of Concerns	The roles are cleanly divided: Merchant sets the price, Subscriber delegates, and the Automation Operator acts as the crucial, time-based third party. This maps perfectly to the required on-chain charge_user instruction authority.
Non-Technical Stories	The user stories are well-written, focusing on user experience ("My wallet prompts me," "A confirmation page appears") rather than technical concepts like PDAs or CPIs, adhering to the INVEST model.
Account Granularity	The account requirements are strong: defining the need for SubscriptionPlan (config), UserSubscription (state), and the Service Authority PDA (signer) provides a direct blueprint for your Anchor code.

---

2. Areas for Refinement and Missing Granularity

While the foundation is strong, the following elements need refinement to ensure a successful Proof-of-Concept. These are the missing pieces that directly translate your functional requirements into API specifications.

A. Missing Granularity in Account Requirements

The current account requirements are at the conceptual level; they need to be translated into specific data fields and primary keys (PDAs).

Missing Detail	Why it Matters	Recommendation for Fix
PDA Seeds (Primary Keys)	We need to know how to deterministically look up these accounts. This maps directly to the #[account(seeds = [..], bump)] macro in Anchor.	Add: Define the PDA seeds for both SubscriptionPlan (e.g., ['plan', merchant_key, plan_id]) and UserSubscription (e.g., ['subscription', subscriber_key, plan_key]).
Subscription Status Fields	The system needs to track payment states beyond just the next date.	Add: UserSubscription needs a status: SubscriptionStatus (e.g., Active, Canceled, Failed) and a failure_count: u8 for retry logic.

Merchant Key Storage	The SubscriptionPlan must link back to the authorized merchant.	Add: SubscriptionPlan must contain the merchant_key: Pubkey of the wallet authorized to update/pause the plan.

B. Unclear/Missing Logic in User Stories & Requirements

The two stories that involve failure or off-chain action need clearer expectations.

Story/Requirement	Missing Clarity	Recommendation for Fix
Automation Operator (charge_user)	The requirement states the Operator must hold a private key, implying Access Control is based on the caller's wallet.	Refine Access Control: Explicitly state the charge_user instruction uses a specific hardcoded public key check (e.g., #[signer] is the Operator's fixed wallet) to authorize the deduction, protecting against external spam.
Subscriber Story (Failure)	("The system sends an on-chain notification or logs a status.") This is too vague for a requirement. Solana programs can't send notifications.	Refine Error Handling: Change the requirement to: The charge_user instruction, upon failure, should log a specific error code (e.g., InsufficientDelegation) and update the UserSubscription status to Failed to signal the Merchant and Subscriber.
Off-Chain Scanning	The Automation Service needs a clear way to handle failures and retries.	Add Requirement: The Automation Service must implement exponential backoff and retry logic for failed charge_user transactions before logging a terminal failure status.