Scheduled Blockchain Payments
Project Description

Costin-Cornel Bosoaga SCPD-1
Dragos-Sebastian Andronic SCPD-1

Project repository

[Link to the project repository]

Project description

Chainvolut is a subscription payments platform on MultiversX that enables users to discover services and pay for subscriptions using EGLD. Service providers can register, list their services, and track subscribers. The platform manages renewals, automatic payments, and cancellations, while eliminating the hassle of working with banks.

Smart contract

- **Endpoints**
  - **registerAsUser**: Registers the caller address as ‘User’;
  - **registerAsProvider**: Registers the caller address as ‘Provider’;
  - **createService**: Allows a Provider to create a new service with a specific name, cost, frequency (in blocks);
  - **deactivateService**: Allows the caller to deactivate a service that they provide.
  - **subscribe**: Allows a user to subscribe to a service (must pay funds > service recurring cost)
  - **topUp**: Allows a subscriber to deposit additional funds into an existing subscription to extend its lifespan.
  - **cancelSubscriptionByUser**: User schedules a cancellation. Subscription remains active until the next payment cycle to prevent immediate termination of paid service.
  - **cancelSubscriptionByProvider**: Schedules a cancellation initiated by the Provider. Similar logic to user cancellation.
  - **triggerPayment**: Checks if the current block nonce >= next payment block. If yes, transfers funds to the vendor and updates the schedule. If funds are low, it cancels the subscription.

    - **finalizeCancellation**: Finalizes cancellation after the block finishes, refunds any remaining balance to the user.

- **Storage Mappers**
    - **services**: Stores Service structs, which contain details about a service, such as name, provider, cost, frequency.
    - **subscriptions**: Stores Subscription structs, which contain details about a subscription such as name, vendor, client, cost, frequency, remaining_balance, next_payment_block.
    - **lastServiceId, lastSubscriptionId**: used as counters for generating IDs.
    - **serviceIds**: A list containing every Service ID created.
    - **providerServices**: Maps a Provider's address to a list of their Service IDs.
    - **userSubscriptions**: Maps a User's address to a list of their Subscription IDs.
    - **providerSubscriptions**: Maps a Vendor's address to a list of incoming Subscription IDs.
    - **serviceSubscriptions**: Maps a Service ID to a list of Subscription IDs associated with it.
    - **userRole:** Maps an address to its role.
    - **cancelRequestedBy, cancelRequestedByIsSet**: Track which address initiated a pending cancellation.

dApp description

The dapp is built with React and TypeScript on top of the MultiversX dApp template and uses @multiversx/sdk-dapp for wallet connection and transaction signing. It implements wallet login and role selection (User or Service Provider), role-based dashboard sections, service creation with name/description/fee/frequency, and subscription management (subscribe, top up, cancel, finalize cancellation). The UI reads data through the contract ABI and shows available services, user subscriptions, provider subscriptions, and service management, depending on the connected role. User actions build and sign contract transactions via the MultiversX SDK, and the UI refreshes on a timer to reflect updated on-chain state without manual reloads.

Scheduler

The scheduler is a Python job that uses the multiversx-sdk core, network providers, and wallet packages. It polls the contract at a fixed interval, reads the subscription state, and triggers scheduled payments only when the current block is greater than or equal to the next payment block. It also finalizes pending cancellations once the current block

reaches the cancel-effective block, so subscriptions transition out of pending automatically. It reads on-chain state through view endpoints and builds, signs, and submits transactions using a PEM signer configured in scheduler/config.py.

Documentation used:

1. https://docs.multiversx.com/
2. https://doc.rust-lang.org/stable/
3. https://react.dev/reference/react
4. https://github.com/multiversx