Your application needs:

1. View-only wallet connection with address input and validation
2. Integration between Streamlit's Python backend and Solana
3. Proper error handling for invalid addresses
4. State management in Streamlit

Let me create the necessary code components to solve these issues.

**1. Updated account.py with Wallet Connection Logic**

First, let's implement the account.py file with proper wallet connection handling:

import streamlit as st

import re

from solana.publickey import PublicKey

from solana.rpc.api import Client

import json

from typing import Optional, Dict, Any

import base58

class WalletConnectionManager:

"""

Manages Solana wallet connections in a view-only capacity.

Handles address validation, connection state, and basic wallet information.

"""

def \_\_init\_\_(self, rpc\_endpoint: str = "https://api.devnet.solana.com"):

"""

Initialize the wallet connection manager.

Args:

rpc\_endpoint: Solana RPC endpoint URL (defaults to devnet)

"""

self.rpc\_endpoint = rpc\_endpoint

self.client = Client(rpc\_endpoint)

self.is\_connected = False

self.wallet\_address = None

self.wallet\_balance = None

self.connection\_error = None

def validate\_solana\_address(self, address: str) -> bool:

"""

Validates if the provided string is a valid Solana address.

Args:

address: The Solana address to validate

Returns:

bool: True if address is valid, False otherwise

"""

# First check: Basic regex pattern for Solana addresses

pattern = r'^[1-9A-HJ-NP-Za-km-z]{32,44}$'

if not re.match(pattern, address):

return False

# Second check: Try to create a PublicKey object

try:

# Verify the address can be properly decoded

decoded = base58.b58decode(address)

if len(decoded) != 32:

return False

# Create a PublicKey object

PublicKey(address)

return True

except (ValueError, TypeError, Exception):

return False

def connect\_wallet(self, address: str) -> Dict[str, Any]:

"""

Connect to a wallet in view-only mode using its address.

Args:

address: The Solana wallet address to connect to

Returns:

Dict containing connection status and information

"""

# Reset connection state

self.is\_connected = False

self.wallet\_address = None

self.wallet\_balance = None

self.connection\_error = None

# Validate the address

if not self.validate\_solana\_address(address):

self.connection\_error = "Invalid Solana address format"

return {

"success": False,

"error": self.connection\_error

}

try:

# Try to connect to RPC endpoint and get account info

self.wallet\_address = address

# Get account balance

response = self.client.get\_balance(PublicKey(address))

if "result" in response and "value" in response["result"]:

lamports = response["result"]["value"]

# Convert lamports to SOL (1 SOL = 1,000,000,000 lamports)

self.wallet\_balance = lamports / 1\_000\_000\_000

self.is\_connected = True

return {

"success": True,

"address": address,

"balance": self.wallet\_balance,

"network": self.rpc\_endpoint

}

else:

self.connection\_error = "Failed to retrieve wallet balance"

return {

"success": False,

"error": self.connection\_error,

"response": response

}

except Exception as e:

self.connection\_error = f"Connection error: {str(e)}"

return {

"success": False,

"error": self.connection\_error

}

def disconnect\_wallet(self) -> Dict[str, Any]:

"""

Disconnect the currently connected wallet.

Returns:

Dict containing disconnection status

"""

self.is\_connected = False

self.wallet\_address = None

self.wallet\_balance = None

self.connection\_error = None

return {

"success": True,

"message": "Wallet disconnected successfully"

}

def get\_wallet\_status(self) -> Dict[str, Any]:

"""

Get the current wallet connection status.

Returns:

Dict containing wallet status information

"""

if self.is\_connected:

return {

"connected": True,

"address": self.wallet\_address,

"balance": self.wallet\_balance,

"network": self.rpc\_endpoint

}

else:

return {

"connected": False,

"error": self.connection\_error

}

def get\_token\_accounts(self) -> Optional[Dict[str, Any]]:

"""

Get token accounts associated with the connected wallet.

Returns:

Dict containing token accounts or None if not connected

"""

if not self.is\_connected or not self.wallet\_address:

return None

try:

response = self.client.get\_token\_accounts\_by\_owner(

PublicKey(self.wallet\_address),

{"programId": PublicKey("TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA")}

)

if "result" in response and "value" in response["result"]:

return {

"success": True,

"token\_accounts": response["result"]["value"]

}

else:

return {

"success": False,

"error": "Failed to retrieve token accounts",

"response": response

}

except Exception as e:

return {

"success": False,

"error": f"Error retrieving token accounts: {str(e)}"

}

def wallet\_connection\_ui():

"""

Creates the wallet connection UI components for Streamlit.

Handles wallet connection state and UI rendering.

"""

# Initialize session state for wallet connection

if "wallet\_manager" not in st.session\_state:

st.session\_state.wallet\_manager = WalletConnectionManager()

# Get the wallet manager from session state

wallet\_manager = st.session\_state.wallet\_manager

# Create two columns for the wallet UI

col1, col2 = st.columns([3, 1])

with col1:

wallet\_address = st.text\_input(

"Enter Solana Wallet Address",

key="wallet\_address\_input",

help="Enter a valid Solana wallet address to connect in view-only mode"

)

with col2:

if wallet\_manager.is\_connected:

if st.button("Disconnect Wallet", key="disconnect\_button"):

result = wallet\_manager.disconnect\_wallet()

if result["success"]:

st.success("Wallet disconnected successfully")

else:

st.error(f"Disconnection failed: {result.get('error', 'Unknown error')}")

else:

if st.button("Connect Wallet", key="connect\_button") and wallet\_address:

with st.spinner("Connecting to wallet..."):

result = wallet\_manager.connect\_wallet(wallet\_address)

if result["success"]:

st.success(f"Connected to wallet: {wallet\_address[:6]}...{wallet\_address[-4:]}")

else:

st.error(f"Connection failed: {result.get('error', 'Unknown error')}")

# Display wallet information if connected

if wallet\_manager.is\_connected:

st.subheader("Wallet Information")

status = wallet\_manager.get\_wallet\_status()

info\_col1, info\_col2 = st.columns(2)

with info\_col1:

st.write("\*\*Network:\*\*", "Solana Devnet" if "devnet" in wallet\_manager.rpc\_endpoint else "Solana Mainnet")

st.write("\*\*Address:\*\*", f"{status['address'][:8]}...{status['address'][-6:]}")

with info\_col2:

st.write("\*\*Balance:\*\*", f"{status['balance']:.9f} SOL")

# Get token accounts

with st.expander("Token Accounts"):

with st.spinner("Loading token accounts..."):

token\_accounts = wallet\_manager.get\_token\_accounts()

if token\_accounts and token\_accounts["success"]:

if len(token\_accounts["token\_accounts"]) > 0:

for i, account in enumerate(token\_accounts["token\_accounts"]):

st.write(f"\*\*Token Account #{i+1}:\*\*", account["pubkey"])

else:

st.info("No token accounts found for this wallet")

else:

error\_msg = token\_accounts.get("error", "Failed to retrieve token accounts") if token\_accounts else "Failed to retrieve token accounts"

st.warning(error\_msg)

return wallet\_manager.is\_connected, wallet\_manager.wallet\_address  
  
**2. Solana Client Implementation**

Now, let's create the solana\_client.py file to handle Solana-specific interactions:

import base58

from solana.rpc.api import Client

from solana.publickey import PublicKey

from solana.rpc.types import TokenAccountOpts

import json

from typing import Dict, Any, List, Optional, Union

import time

class SolanaClient:

"""

Client for interacting with the Solana blockchain.

Provides methods for account information, transactions, and token data.

"""

def \_\_init\_\_(self, rpc\_endpoint: str = "https://api.devnet.solana.com"):

"""

Initialize the Solana client with an RPC endpoint.

Args:

rpc\_endpoint: Solana RPC endpoint URL (defaults to devnet)

"""

self.rpc\_endpoint = rpc\_endpoint

self.client = Client(rpc\_endpoint)

self.connection\_status = self.\_check\_connection()

def \_check\_connection(self) -> Dict[str, Any]:

"""

Check if the connection to the Solana RPC endpoint is working.

Returns:

Dict containing connection status information

"""

try:

# Get the version to test connection

response = self.client.get\_version()

if "result" in response:

return {

"connected": True,

"version": response["result"],

"endpoint": self.rpc\_endpoint

}

else:

return {

"connected": False,

"error": "Unexpected response format",

"endpoint": self.rpc\_endpoint

}

except Exception as e:

return {

"connected": False,

"error": str(e),

"endpoint": self.rpc\_endpoint

}

def get\_balance(self, address: str) -> Dict[str, Any]:

"""

Get the SOL balance for a wallet address.

Args:

address: Solana wallet address

Returns:

Dict containing balance information

"""

try:

pubkey = PublicKey(address)

response = self.client.get\_balance(pubkey)

if "result" in response and "value" in response["result"]:

lamports = response["result"]["value"]

sol\_balance = lamports / 1\_000\_000\_000 # Convert lamports to SOL

return {

"success": True,

"balance\_lamports": lamports,

"balance\_sol": sol\_balance,

"address": address

}

else:

return {

"success": False,

"error": "Failed to retrieve balance",

"response": response,

"address": address

}

except Exception as e:

return {

"success": False,

"error": f"Error retrieving balance: {str(e)}",

"address": address

}

def get\_account\_info(self, address: str) -> Dict[str, Any]:

"""

Get detailed account information for a wallet address.

Args:

address: Solana wallet address

Returns:

Dict containing account information

"""

try:

pubkey = PublicKey(address)

response = self.client.get\_account\_info(pubkey)

if "result" in response and "value" in response["result"]:

account\_data = response["result"]["value"]

return {

"success": True,

"account\_data": account\_data,

"address": address

}

else:

return {

"success": False,

"error": "Failed to retrieve account information",

"response": response,

"address": address

}

except Exception as e:

return {

"success": False,

"error": f"Error retrieving account information: {str(e)}",

"address": address

}

def get\_token\_accounts(self, owner\_address: str) -> Dict[str, Any]:

"""

Get all token accounts owned by the specified address.

Args:

owner\_address: Solana wallet address

Returns:

Dict containing token account information

"""

try:

pubkey = PublicKey(owner\_address)

token\_program\_id = PublicKey("TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA")

opts = TokenAccountOpts(program\_id=token\_program\_id)

response = self.client.get\_token\_accounts\_by\_owner(pubkey, opts)

if "result" in response and "value" in response["result"]:

token\_accounts = response["result"]["value"]

processed\_accounts = []

for account in token\_accounts:

try:

account\_data = base58.b58decode(account["account"]["data"][0])

# Process token account data if needed

processed\_accounts.append({

"pubkey": account["pubkey"],

"account": account["account"]

})

except Exception as e:

# Skip accounts that can't be processed

continue

return {

"success": True,

"token\_accounts": processed\_accounts,

"address": owner\_address

}

else:

return {

"success": False,

"error": "Failed to retrieve token accounts",

"response": response,

"address": owner\_address

}

except Exception as e:

return {

"success": False,

"error": f"Error retrieving token accounts: {str(e)}",

"address": owner\_address

}

def get\_recent\_transactions(self, address: str, limit: int = 10) -> Dict[str, Any]:

"""

Get recent transactions for a wallet address.

Args:

address: Solana wallet address

limit: Maximum number of transactions to return

Returns:

Dict containing transaction information

"""

try:

pubkey = PublicKey(address)

# Get signatures for the address

response = self.client.get\_signatures\_for\_address(pubkey, limit=limit)

if "result" in response:

signatures = response["result"]

transactions = []

for sig\_info in signatures:

tx\_details = {

"signature": sig\_info["signature"],

"slot": sig\_info["slot"],

"err": sig\_info.get("err", None),

"memo": sig\_info.get("memo", None),

"block\_time": sig\_info.get("blockTime", None),

"confirmation\_status": sig\_info.get("confirmationStatus", None)

}

transactions.append(tx\_details)

return {

"success": True,

"transactions": transactions,

"address": address

}

else:

return {

"success": False,

"error": "Failed to retrieve transactions",

"response": response,

"address": address

}

except Exception as e:

return {

"success": False,

"error": f"Error retrieving transactions: {str(e)}",

"address": address

}

def is\_valid\_solana\_address(self, address: str) -> bool:

"""

Validate if a string is a valid Solana address.

Args:

address: String to validate as Solana address

Returns:

bool: True if valid, False otherwise

"""

try:

# Check if the address can be decoded as base58 and has correct length

decoded = base58.b58decode(address)

if len(decoded) != 32:

return False

# Try to create PublicKey object

PublicKey(address)

return True

except Exception:

return False

**3. Main Application File**

Now, let's create the main Streamlit application file that integrates everything:

import streamlit as st

import sys

import os

from components.account import wallet\_connection\_ui

from solana\_client import SolanaClient

import time

import json

# Set page configuration

st.set\_page\_config(

page\_title="Solana Wallet Viewer",

page\_icon="💰",

layout="wide",

initial\_sidebar\_state="expanded"

)

# Initialize session state variables

if "initialized" not in st.session\_state:

st.session\_state.initialized = True

st.session\_state.connected = False

st.session\_state.wallet\_address = None

st.session\_state.connection\_time = None

st.session\_state.solana\_client = SolanaClient(rpc\_endpoint="https://api.devnet.solana.com")

def check\_rpc\_connection():

"""Check if the connection to Solana RPC is working"""

client = st.session\_state.solana\_client

status = client.connection\_status

if status["connected"]:

st.sidebar.success(f"✅ Connected to Solana {status['endpoint']}")

st.sidebar.write(f"Version: {status['version'].get('solana-core', 'Unknown')}")

else:

st.sidebar.error(f"❌ Failed to connect to {status['endpoint']}")

st.sidebar.write(f"Error: {status.get('error', 'Unknown error')}")

def display\_wallet\_transactions(wallet\_address):

"""Display recent transactions for the connected wallet"""

if not wallet\_address:

return

client = st.session\_state.solana\_client

with st.spinner("Loading recent transactions..."):

tx\_data = client.get\_recent\_transactions(wallet\_address)

if tx\_data["success"] and tx\_data["transactions"]:

st.subheader("Recent Transactions")

for tx in tx\_data["transactions"]:

with st.expander(f"Transaction: {tx['signature'][:10]}...{tx['signature'][-6:]}"):

col1, col2 = st.columns(2)

with col1:

st.write("\*\*Signature:\*\*", tx["signature"])

st.write("\*\*Slot:\*\*", tx["slot"])

if tx["block\_time"]:

st.write("\*\*Time:\*\*", time.strftime('%Y-%m-%d %H:%M:%S', time.localtime(tx["block\_time"])))

with col2:

status\_color = "🟢" if not tx["err"] else "🔴"

st.write(f"\*\*Status:\*\* {status\_color} {'Success' if not tx['err'] else 'Failed'}")

st.write("\*\*Confirmation:\*\*", tx["confirmation\_status"] or "Unknown")

if tx["err"]:

st.write("\*\*Error:\*\*", str(tx["err"]))

if tx["memo"]:

st.write("\*\*Memo:\*\*", tx["memo"])

elif tx\_data["success"]:

st.info("No recent transactions found for this wallet")

else:

st.warning(f"Failed to load transactions: {tx\_data.get('error', 'Unknown error')}")

def display\_account\_info(wallet\_address):

"""Display detailed account information for the connected wallet"""

if not wallet\_address:

return

client = st.session\_state.solana\_client

with st.spinner("Loading account information..."):

account\_info = client.get\_account\_info(wallet\_address)

if account\_info["success"] and account\_info["account\_data"]:

st.subheader("Account Information")

data = account\_info["account\_data"]

col1, col2 = st.columns(2)

with col1:

st.write("\*\*Executable:\*\*", data.get("executable", False))

st.write("\*\*Lamports:\*\*", data.get("lamports", 0))

st.write("\*\*Rent Epoch:\*\*", data.get("rentEpoch", 0))

with col2:

if "owner" in data:

st.write("\*\*Owner Program:\*\*", data["owner"])

if "data" in data and isinstance(data["data"], list) and len(data["data"]) > 0:

st.write("\*\*Data Size:\*\*", len(data["data"][0]) if data["data"][0] else "0 bytes")

elif account\_info["success"]:

st.info("No detailed account information available")

else:

st.warning(f"Failed to load account information: {account\_info.get('error', 'Unknown error')}")

def main():

"""Main application entry point"""

st.title("Solana Wallet Viewer")

st.write("Connect to a Solana wallet in view-only mode to see balance and activity")

# Display Solana RPC connection status in sidebar

st.sidebar.title("Network Status")

check\_rpc\_connection()

# Network selection in sidebar

st.sidebar.subheader("Network Settings")

network = st.sidebar.radio(

"Select Solana Network",

options=["Devnet", "Mainnet Beta"],

index=0, # Default to Devnet

key="network\_selection"

)

# Update RPC endpoint if network changes

current\_endpoint = st.session\_state.solana\_client.rpc\_endpoint

new\_endpoint = "https://api.devnet.solana.com" if network == "Devnet" else "https://api.mainnet-beta.solana.com"

if current\_endpoint != new\_endpoint:

st.session\_state.solana\_client = SolanaClient(rpc\_endpoint=new\_endpoint)

st.experimental\_rerun()

# Display wallet connection UI

st.subheader("Wallet Connection")

connected, wallet\_address = wallet\_connection\_ui()

# Update session state

st.session\_state.connected = connected

st.session\_state.wallet\_address = wallet\_address

if connected and wallet\_address:

# Store connection time

if not st.session\_state.connection\_time:

st.session\_state.connection\_time = time.time()

# Show time since connection

elapsed = time.time() - st.session\_state.connection\_time

st.sidebar.write(f"Connected for: {int(elapsed)} seconds")

# Display tabs for different wallet information

tab1, tab2, tab3 = st.tabs(["Transactions", "Account Details", "Token Accounts"])

with tab1:

display\_wallet\_transactions(wallet\_address)

with tab2:

display\_account\_info(wallet\_address)

with tab3:

st.subheader("Token Accounts")

client = st.session\_state.solana\_client

with st.spinner("Loading token accounts..."):

token\_data = client.get\_token\_accounts(wallet\_address)

if token\_data["success"] and token\_data["token\_accounts"]:

for i, account in enumerate(token\_data["token\_accounts"]):

st.write(f"\*\*Token Account #{i+1}:\*\*", account["pubkey"])

st.json(account)

st.divider()

elif token\_data["success"]:

st.info("No token accounts found for this wallet")

else:

st.warning(f"Failed to load token accounts: {token\_data.get('error', 'Unknown error')}")

else:

# Reset connection time if disconnected

st.session\_state.connection\_time = None

# Show instructions

st.info("Enter a valid Solana wallet address above and click 'Connect Wallet' to view wallet information.")

st.write("Example devnet address: 83astBRguLMdt2h5U1Tpdq5tjFoJ6noeGwaY3mDLVcri")

if \_\_name\_\_ == "\_\_main\_\_":

try:

main()

except Exception as e:

st.error(f"An unexpected error occurred: {str(e)}")

st.exception(e)  
  
**4. Server Configuration**

Finally, let's create a configuration file to ensure the server runs on 0.0.0.0 and port 5000:

#!/bin/bash

# Run Streamlit server on 0.0.0.0:5000

# Make sure Python environment is activated

echo "Starting Solana Wallet Viewer application..."

# Set Streamlit server configuration

export STREAMLIT\_SERVER\_PORT=5000

export STREAMLIT\_SERVER\_ADDRESS=0.0.0.0

export STREAMLIT\_SERVER\_HEADLESS=true

# Run the Streamlit application

streamlit run app.py --server.port 5000 --server.address 0.0.0.0  
  
**Solana Wallet Viewer for Streamlit**

I've created a comprehensive solution for your Solana web application with view-only wallet connection functionality. Let me explain the key components and how they work together:

**Solution Overview**

The solution I've provided addresses all your requirements:

1. **View-only wallet connection** with proper address validation
2. **Streamlit integration** for the frontend
3. **Solana client implementation** for blockchain interactions
4. **Error handling** for invalid addresses and connection failures
5. **State management** using Streamlit's session state

Let me walk you through each component and explain how they work together:

**1. Wallet Connection Manager (account.py)**

The account.py file contains a WalletConnectionManager class that handles:

* Validating Solana addresses using both regex patterns and the Solana SDK
* Connecting to wallets in view-only mode
* Managing connection state
* Retrieving wallet balances and token accounts

The wallet\_connection\_ui() function creates the Streamlit UI components for wallet connection, including:

* Input field for wallet addresses
* Connect/Disconnect buttons
* Wallet information display
* Token account information

The address validation uses multiple checks:

1. A regex pattern to verify basic format
2. Attempting to decode using base58
3. Creating a PublicKey object from the address

**2. Solana Client (solana\_client.py)**

The SolanaClient class provides a comprehensive interface to the Solana blockchain:

* Connecting to different Solana networks (devnet/mainnet)
* Retrieving account balances in both lamports and SOL
* Fetching account information and token accounts
* Getting recent transactions
* Validating Solana addresses

It includes robust error handling that returns detailed information about any failures, making it easier to debug issues.

**3. Main Application (app.py)**

The main Streamlit application integrates these components:

* Sets up the page configuration and UI layout
* Initializes session state variables
* Manages RPC connection status
* Provides network selection (Devnet/Mainnet)
* Organizes wallet information in tabs (Transactions, Account Details, Token Accounts)
* Handles errors gracefully with appropriate user feedback

**4. Server Configuration (run\_server.sh)**

The server configuration script ensures your application:

* Binds to 0.0.0.0 as required
* Uses port 5000
* Sets appropriate Streamlit server parameters

**Key Features and Implementation Details**

**Address Validation**

The solution implements thorough Solana address validation:

python

def validate\_solana\_address(self, address: str) -> bool:

*# Basic regex pattern check*

pattern = r'^[1-9A-HJ-NP-Za-km-z]{32,44}$'

if not re.match(pattern, address):

return False

*# Try to create a PublicKey object and verify proper decoding*

try:

decoded = base58.b58decode(address)

if len(decoded) != 32:

return False

PublicKey(address)

return True

except (ValueError, TypeError, Exception):

return False

This two-step validation ensures that addresses are not just syntactically correct but also properly formatted for the Solana blockchain.

**Error Handling**

Error handling is implemented at multiple levels:

1. **Function-level error handling**: Each function catches exceptions and returns structured error information
2. **UI-level feedback**: Error messages are displayed to users in context
3. **Application-level try/except**: The main function is wrapped in a try/except block to catch any unhandled exceptions

**Error Handling (continued)**

For example, in the wallet connection function, errors are properly captured and displayed to the user:

python

result = wallet\_manager.connect\_wallet(wallet\_address)

if result["success"]:

st.success(f"Connected to wallet: {wallet\_address[:6]}...{wallet\_address[-4:]}")

else:

st.error(f"Connection failed: {result.get('error', 'Unknown error')}")

This pattern is used throughout the application to provide clear feedback when operations fail.

**State Management**

The application uses Streamlit's session state to maintain wallet connection information across interactions:

python

*# Initialize session state variables*

if "initialized" not in st.session\_state:

st.session\_state.initialized = True

st.session\_state.connected = False

st.session\_state.wallet\_address = None

st.session\_state.connection\_time = None

st.session\_state.solana\_client = SolanaClient(rpc\_endpoint="https://api.devnet.solana.com")

This ensures that wallet connection state persists even when components are reloaded or the user interacts with different parts of the UI.

**Network Configuration**

The application allows users to switch between Solana Devnet and Mainnet Beta:

python

network = st.sidebar.radio(

"Select Solana Network",

options=["Devnet", "Mainnet Beta"],

index=0, *# Default to Devnet*

key="network\_selection"

)

*# Update RPC endpoint if network changes*

current\_endpoint = st.session\_state.solana\_client.rpc\_endpoint

new\_endpoint = "https://api.devnet.solana.com" if network == "Devnet" else "https://api.mainnet-beta.solana.com"

if current\_endpoint != new\_endpoint:

st.session\_state.solana\_client = SolanaClient(rpc\_endpoint=new\_endpoint)

st.experimental\_rerun()

This allows for testing on Devnet before deploying to Mainnet.

**Integration with Your Existing Code**

To integrate this solution with your existing UniversalWalletProvider.tsx and WalletContext.tsx:

1. Your React components can communicate with the Streamlit backend using API endpoints
2. The WalletConnectionManager in account.py handles the backend wallet operations
3. Token accounts and transactions are handled by the SolanaClient class

The solution is designed to work with your existing components while providing the additional functionality you need.

**Running the Application**

To run the application:

1. Make sure you have the required packages installed:

bash

pip install streamlit solana base58

1. Set up your project directory structure:
2. your\_project/
3. ├── app.py
4. ├── solana\_client.py
5. ├── run\_server.sh
6. └── components/

└── account.py

1. Make the run script executable:

bash

chmod +x run\_server.sh

1. Run the application:

bash

./run\_server.sh

This will start the Streamlit server on 0.0.0.0:5000 as required.

**Troubleshooting Common Issues**

If you encounter issues with the wallet connection:

1. **Verify RPC endpoints**: Make sure the Solana RPC endpoints are accessible from your server
2. **Check wallet address format**: Ensure addresses are properly formatted (see examples in app.py)
3. **Network issues**: The application shows RPC connection status in the sidebar

If an error occurs during RPC communication, the application will display specific error messages to help with debugging.

**Security Considerations**

Since this is a view-only wallet implementation:

1. No private keys are stored or required
2. All connections use public addresses only
3. No transaction signing capabilities are included

This makes the application secure for viewing wallet information without risking funds.