Task 1 : Address Lookup:

Allow search for any Ripple address and display full transaction history

Context : The user enters a valid Ripple address (starting with r). The system connects to the Ripple testnet and:

1.Fetches the latest transactions made by that address

2.Displays important details like:

- Transaction type
- Amount sent or received
- Sender and receiver addresses
- Transaction result
- Date and time of the transaction
- Transaction Fee
- Ledger Index

Code :

```
import xrpl from "xrpl";

async function main() {
  const client = new xrpl.Client("wss://s.altnet.rippletest.net:51233");
  await client.connect();

  async function getAccountTransactions(address) {
    const txs = await client.request({
      command: "account_tx", // Ripple command to get transaction history
      account: address, // Ripple address
      ledger_index_min: -1, // Start from the earliest ledger
      ledger_index_max: -1, // Up to the latest ledger
      limit: 2, // number of transactions to fetch
    });

    console.log(`Transaction History for ${address}`);
    const transactions = txs.result.transactions;
    if (transactions.length === 0) {
      console.log("No transactions found.");
    } else {
      transactions.forEach((tx, index) => {
        console.log(`\n#${index + 1}`);
        console.log(" Type:", tx.tx_json.TransactionType);
        console.log(" Hash:", tx.hash);
        console.log(
          " Amount:",
          xrpl.dropsToXrp(tx.meta.delivered_amount) + " XRP"
        );
        console.log(" From:", tx.tx_json.Account);
        console.log(" To:", tx.tx_json.Destination);
        const rippleDate = new Date(
          (tx.tx_json.date + 946684800) * 1000
        ).toLocaleString("en-IN", {
```

```
          timeZone: "Asia/Kolkata",
          hour12: true,
        });
        console.log(" Date:", rippleDate);
        const feeXRP = xrpl.dropsToXrp(tx.tx_json.Fee);
        const Ledger_Index = tx.tx_json.ledger_index;
        console.log(" Fee:", feeXRP + " XRP");
        console.log(" Ledger Index:", Ledger_Index);
        console.log(" Result:", tx.meta.TransactionResult);
      });
    }
  }

  await getAccountTransactions("rPT1Sjq2YGrBMTttX4GZHjKu9dyfzbpAYe");

  await client.disconnect();
}

main();
```

Output :

```
Transaction History for rPT1Sjq2YGrBMTttX4GZHjKu9dyfzbpAYe

#1
 Type: Payment
 Hash: EF978AB0D82AD81804774EC454593598DFDB583BD7A6E16806FDC982A3D74E41
 Amount: 10 XRP
 From: rPT1Sjq2YGrBMTttX4GZHjKu9dyfzbpAYe
 To: rHmLhQ6WQxZYjB2uVJHzpYjVubcU7DCevW
 Date: 17/6/2025, 6:12:30 pm
 Fee: 0.000012 XRP
 Ledger Index: 8170533
 Result: tesSUCCESS

#2
 Type: Payment
 Hash: 8A40CC4F054E70CF9FA6CAC95C09D40356779C759BD0634FBA7F58470BA1857A
 Amount: 10 XRP
 From: rPT1Sjq2YGrBMTttX4GZHjKu9dyfzbpAYe
 To: rJat93t547tBQnu14Ary28a8zHN9p9sWKj
 Date: 17/6/2025, 6:12:30 pm
 Fee: 0.000012 XRP
 Ledger Index: 8170533
 Result: tesSUCCESS
```

Task 2 : FLO Private Key Integration:

Enable sending of XRP using a valid FLO blockchain private key or using Ripple address private key of the sender

Context : The user can do XRP transfers using a private key from either:

- The **FLO blockchain** (in WIF format)
- A standard **Ripple secret key** (starts with s...)

Code :

```
import bs58 from "bs58";
import xrpl from "xrpl";
function floWIFtoRippleWallet(floWIF) {
  try {
    // Decode Base58 WIF
    const decoded = bs58.decode(floWIF);

    // Remove 1-byte prefix and 4-byte checksum
    let keyBytes = decoded.slice(1, -4); // [1] network prefix, [last 4] checksum

    // Remove last byte (0x01) if compressed
    if (keyBytes.length === 33 && keyBytes[32] === 0x01) {
      keyBytes = keyBytes.slice(0, 32);
    }

    // Create entropy buffer
    const entropy = Uint8Array.from(keyBytes);

    // Generate Ripple wallet from entropy
    const wallet = xrpl.Wallet.fromEntropy(entropy, { algorithm: "secp256k1" });

    return wallet;
  } catch (err) {
    console.error( err.message);
    return null;
  }
}
const rippleWallet =
floWIFtoRippleWallet("L1Y7sMZ7cLq2ZQvN9txRzqNE7wUvhTejXStMH36TSqAF2FS1vyyH");

if (rippleWallet) {
  console.log("Ripple Wallet from FLO Key:");
  console.log("Address:", rippleWallet.classicAddress);
  console.log("Public Key:", rippleWallet.publicKey);
  console.log("Private Key:", rippleWallet.privateKey);
  console.log("Secret Seed:", rippleWallet.seed);
}
```

Output :

```
Ripple Wallet from FLO Key:
Address: rM5PJ846Mdnu7cHrFPSd3KixaXS5ievbLa
Public Key: 03C41898299198805BE719B4EAD38087CA1D7411A09EEC90AC50290D80A0A322AD
Private Key: 00177AC32E43BB3DC9A06F1B17128F9BB0FBAA48DAE73BD9203DD5FD6E1994303D
Secret Seed: shVP5eeXiA1sAx7H2KC5vqaD7khbK
```

Task 3 : Multi-Chain Address Generation:

On creating a new Ripple address, automatically generate and display:

a) Equivalent FLO address

b) Equivalent Bitcoin address

c) Associated private keys for all three

Context : It helps to **generate a Ripple wallet** and simultaneously create **equivalent addresses** for:

- FLO
- Bitcoin
- Along with their respective **private keys**

Code :

```
import xrpl from "xrpl";
import * as bitcoin from "bitcoinjs-lib";
import * as ECPairFactory from "ecpair";
import * as ecc from "tiny-secp256k1";

const ECPair = ECPairFactory.ECPairFactory(ecc);

// FLO address generator with FLO prefix (0x23)
function getFloAddressFromPrivKeyHex(privKeyHex) {
  const keyPair = ECPair.fromPrivateKey(Buffer.from(privKeyHex, "hex"), { compressed: true
});
  const { address } = bitcoin.payments.p2pkh({
  pubkey: Buffer.from(keyPair.publicKey),
  network: { pubKeyHash: 0x23, wif: 0x80 }
});

  return address;
}

function generateMultiChainWallet() {
  // Generate Ripple wallet
  const entropy = crypto.getRandomValues(new Uint8Array(16));
  const rippleWallet = xrpl.Wallet.fromEntropy(entropy, { algorithm: "secp256k1" });

  // Extract private key hex (skip 0x00 prefix)
  const privKeyHex = rippleWallet.privateKey.slice(2);
  const privKeyBuffer = Buffer.from(privKeyHex, "hex");

  // Generate Bitcoin key pair and address
  const btcKeyPair = ECPair.fromPrivateKey(privKeyBuffer, { compressed: true });
  const btcAddress = bitcoin.payments.p2pkh({ pubkey: Buffer.from(btcKeyPair.publicKey)
}).address;
```

```
  const btcWIF = btcKeyPair.toWIF();

  // Generate FLO address from same private key
  const floAddress = getFloAddressFromPrivKeyHex(privKeyHex);

  return {
    ripple: {
      address: rippleWallet.classicAddress,
      privateKey: rippleWallet.privateKey,
      seed: rippleWallet.seed
    },
    bitcoin: {
      address: btcAddress,
      privateKey: btcWIF
    },
    flo: {
      address: floAddress,
      privateKey: btcWIF
    }
  };
}

const wallet = generateMultiChainWallet();

console.log(" Ripple Address:", wallet.ripple.address);
console.log(" Ripple Private Key:", wallet.ripple.privateKey);
console.log(" Ripple Seed:", wallet.ripple.seed);
console.log("\n Bitcoin Address:", wallet.bitcoin.address);
console.log(" BTC Private Key (WIF):", wallet.bitcoin.privateKey);
console.log("\n FLO Address:", wallet.flo.address);
console.log(" FLO Private Key (WIF):", wallet.flo.privateKey);
```

Output :

```
Ripple Address: rnHmkBSfR6rbqFXMXd6iNTZFGg4CJ6q4BM
Ripple Private Key: 006F828079A37BD294B1FACC7EF1637A53587FB0D2465C29A32C452763C21C9E56
Ripple Seed: ssbxmKzrLjGzkfhCmm2X4B2afwZdZ

Bitcoin Address: 15HmkBS7Rh1bqpXMXdhrETZpGgNfJhqNBM
BTC Private Key (WIF): KzxUHjZLkwtNwF5iU6j6BKRsBAoCJGQhUiWpeYwDVbU26bGK8wkW

FLO Address: FA7tCysCH1EGTzQPPKN1Cr6MJLPgBZKH8M
FLO Private Key (WIF): KzxUHjZLkwtNwF5iU6j6BKRsBAoCJGQhUiWpeYwDVbU26bGK8wkW
```

Task 4 : Private Key-Based Address Recovery:

Derive the original Ripple address from a valid FLO, Bitcoin, or Ripple private key.

Context : Helps a user to **recover their original Ripple address** using any of the following types of private keys:

- A **Ripple secret seed** (starts with s)
- A **FLO private key** in **WIF format**
- A **Bitcoin private key** in **WIF format**

Code :

```
import xrpl from "xrpl";
import bs58 from "bs58";


function recoverRippleWallet(inputKey) {
  try {
    let wallet;

    if (inputKey.startsWith("s")) {
      // Ripple seed
      wallet = xrpl.Wallet.fromSeed(inputKey);
    } else {
      // FLO/BTC WIF
      const decoded = bs58.decode(inputKey);

      let keyBytes = decoded.slice(1, -4); // Remove version and checksum

      // For compressed WIF, remove last byte (0x01)
      if (keyBytes.length === 33 && keyBytes[32] === 0x01)
        keyBytes = keyBytes.slice(0, 32);

      wallet = xrpl.Wallet.fromEntropy(Uint8Array.from(keyBytes), { algorithm: "secp256k1"
});
    }

    return {
      rippleAddress: wallet.classicAddress,
      publicKey: wallet.publicKey,
      privateKey: wallet.privateKey,
      seed: wallet.seed,
    };
  } catch (err) {
    console.error(err.message);
    return null;
  }
}


const inputKey = "rPT1Sjq2YGrBMTttX4GZHjKu9dyfzbpAYe";
const result = recoverRippleWallet(inputKey);

if (result) {
  console.log(" Ripple Address:", result.rippleAddress);
  console.log(" Public Key:", result.publicKey);
```

```
  console.log(" Private Key:", result.privateKey);
  console.log(" Ripple Seed:", result.seed);
}
```

Output :

```
Ripple Address: rnHmkBSfR6rbqFXMXd6iNTZFGg4CJ6q4BM
Ripple Private Key: 006F828079A37BD294B1FACC7EF1637A53587FB0D2465C29A32C452763C21C9E56
Ripple Seed: ssbxmKzrLjGzkfhCmm2X4B2afwZdZ

Bitcoin Address: 15HmkBS7Rh1bqpXMXdhrETZpGgNfJhqNBM
BTC Private Key (WIF): KzxUHjZLkwtNwF5iU6j6BKRsBAoCJGQhUiWpeYwDVbU26bGK8wkW

FLO Address: FA7tCysCH1EGTzQPPKN1Cr6MJLPgBZKH8M
FLO Private Key (WIF): KzxUHjZLkwtNwF5iU6j6BKRsBAoCJGQhUiWpeYwDVbU26bGK8wkW
```

Task 5 : Balance Retrieval:

Show XRP balance for any address, using:

a) Ripple blockchain address, or

b) Corresponding FLO / Bitcoin private keys

Context : Helps the user to **check the XRP balance** of a wallet using any of the following:

- A **Ripple address** (starting with r)
- A **Ripple seed** (starting with s)
- A **FLO or Bitcoin private key** in **WIF format**

Code :

```
import xrpl from "xrpl";
import bs58 from "bs58";

//Convert WIF (FLO/BTC) to Ripple wallet

function convertWIFtoRippleWallet(wif) {
  const decoded = bs58.decode(wif);
  let keyBytes = decoded.slice(1, -4);
  if (keyBytes.length === 33 && keyBytes[32] === 0x01)
    keyBytes = keyBytes.slice(0, 32);
  return xrpl.Wallet.fromEntropy(Uint8Array.from(keyBytes), {
    algorithm: "secp256k1",
  });
}

//Ripple wallet or address from input

function getRippleAddress(input) {
  if (input.startsWith("r")) return input;
  if (input.startsWith("s")) return xrpl.Wallet.fromSeed(input).classicAddress;
  try {
    return convertWIFtoRippleWallet(input).classicAddress;
```

```
  } catch {
    return null;
  }
}

//Get XRP balance from address or private key

async function getXRPBalance(input) {
  const address = getRippleAddress(input);
  if (!address) {
    console.error("Invalid input. Must be Ripple address or private key.");
    return;
  }

  const client = new xrpl.Client("wss://s.altnet.rippletest.net:51233");
  await client.connect();

  try {
    const response = await client.request({
      command: "account_info",
      account: address,
      ledger_index: "validated",
    });

    const balanceDrops = response.result.account_data.Balance;
    const balanceXRP = xrpl.dropsToXrp(balanceDrops);

    console.log(`Balance for ${address}: ${balanceXRP} XRP`);
  } catch (err) {
    console.error("Failed to fetch balance:", err.message);
  } finally {
    await client.disconnect();
  }
}

const input = "sEd7J8AQD5JjdFqFFZrziHvA3AwXcP5";
getXRPBalance(input);
```

Output:

```
(Use `node --trace-warnings ...` to show where the warning was created)
Balance for rGmUSnRJduJz6x9wGhZ9NdwPJyiewxvtZd: 9.999976 XRP
```

Task 6 : Token Transfer:

Enable sending of XRP using:

a) Ripple private key, or b) Its corresponding/equivalent FLO and Bitcoin private keys

Context : It helps a user to **send XRP tokens** using either:

- A **Ripple private key** (seed format, starts with s)
- Or a **corresponding private key** from the **FLO or Bitcoin blockchains** (in WIF format)

Code :

```
import xrpl from "xrpl";
import bs58 from "bs58";

// Convert FLO/BTC WIF to Ripple wallet

function convertWIFtoRippleWallet(wif) {
  const decoded = bs58.decode(wif);
  let keyBytes = decoded.slice(1, -4);
  if (keyBytes.length === 33 && keyBytes[32] === 0x01)
    keyBytes = keyBytes.slice(0, 32);
  return xrpl.Wallet.fromEntropy(Uint8Array.from(keyBytes), {
    algorithm: "secp256k1",
  });
}

// Recover wallet from private key (Ripple or WIF)
function getWalletFromPrivateKey(inputKey) {
  if (inputKey.startsWith("s")) return xrpl.Wallet.fromSeed(inputKey);
  return convertWIFtoRippleWallet(inputKey);
}

// Send XRP from private key to destination

async function sendXRP(senderPrivKey, destination, amountXRP) {
  const wallet = getWalletFromPrivateKey(senderPrivKey);
  const client = new xrpl.Client("wss://s.altnet.rippletest.net:51233");
  await client.connect();

  try {
    const tx = {
      TransactionType: "Payment",
      Account: wallet.classicAddress,
      Destination: destination,
      Amount: xrpl.xrpToDrops(amountXRP.toString()),
    };

    const prepared = await client.autofill(tx);
    const signed = wallet.sign(prepared);
    const result = await client.submitAndWait(signed.tx_blob);

    console.log(" TX Hash:", signed.hash);
    console.log(" From:", wallet.classicAddress);
    console.log(" To:", destination);
```

```
    console.log(" Amount:", amountXRP, "XRP");
    const rippleDate = new Date(
      (result.result.tx_json.date + 946684800) * 1000
    ).toLocaleString("en-IN", {
      timeZone: "Asia/Kolkata",
      hour12: true,
    });
    console.log(" Date:", rippleDate);

    const Ledger_Index = result.result.tx_json.ledger_index;
    console.log(" Fee:", xrpl.dropsToXrp(result.result.tx_json.Fee), "XRP");
    console.log(" Ledger Index:", Ledger_Index);
    console.log(" Result:", result.result.meta.TransactionResult);
  } catch (err) {
    console.error("Transaction failed:", err.message);
  } finally {
    await client.disconnect();
  }
}

const senderKey = "sEd7J8AQD5JjdFqFFZrziHvA3AwXcP5";
const to = "rPT1Sjq2YGrBMTttX4GZHjKu9dyfzbpAYe";
const amount = 1;

sendXRP(senderKey, to, amount);
```

Output :

```
TX Hash: 87F22C566F65B693C9C61347A8BA1CF303324FFDB8F0220BA48F01A697731096
From: rGmUSnRJduJz6x9wGhZ9NdwPJyiewxvtZd
To: rPT1Sjq2YGrBMTttX4GZHjKu9dyfzbpAYe
Amount: 1 XRP
Date: 18/6/2025, 12:13:22 am
Fee: 0.000012 XRP
Ledger Index: 8176908
Result: tesSUCCESS
```