



NUS Fintech Summit 2026

Pre-Hackathon Workshop

The Challenge

Build an MVP that leverages the XRP Ledger's core features to **solve a real-world problem**. We're seeking innovative solutions with a variety of use-cases. If you don't know where to start, we're especially interested in applications around payments, Real-World Asset (RWA) tokenization, RLUSD integration, credentials and/or decentralized identity (DID).

A successful project will solve a real-world financial problem using XRPL. Features could include:

- Credentials/DID-integrated fintech identity flows
- Permissioned flows using on chain credentials
- RLUSD-based apps
- SDKs for other developers to easily integrate XRPL features like RLUSD, escrows, and DIDs into their app
- Payment apps, microfinance, Real-World Asset (RWA) Tokenization
- Cross-chain integration with EVM sidechain

Prizes

To compete, you must submit a working, testable MVP that is publicly available on GitHub with a detailed README. Several winning projects will be awarded a track prize:

5000 SGD for 1st

3000 SGD for 2nd

1000 SGD for 3rd

with 1-2 additional **500 SGD bounties for the best developer feedback.**

Developer Feedback

Submit at the [Developer Feedback Portal](#)

Submit

- Typo in the docs
- Confusing error message
- Broken faucet
- Additional tutorials needed

The top teams that we receive the best developer feedback from (most + highest quality) will receive a 500 SGD award 🏆 at the end of the program.



The Ripple Judges



Dr. Nasri Bin Najib
Staff Partner Engineer



Tatsuya Kohrog
Senior Ecosystem Growth Manager



Chris Howes
Director, Partner Engineering



Emma Nasseri
Program Manager, UBRI



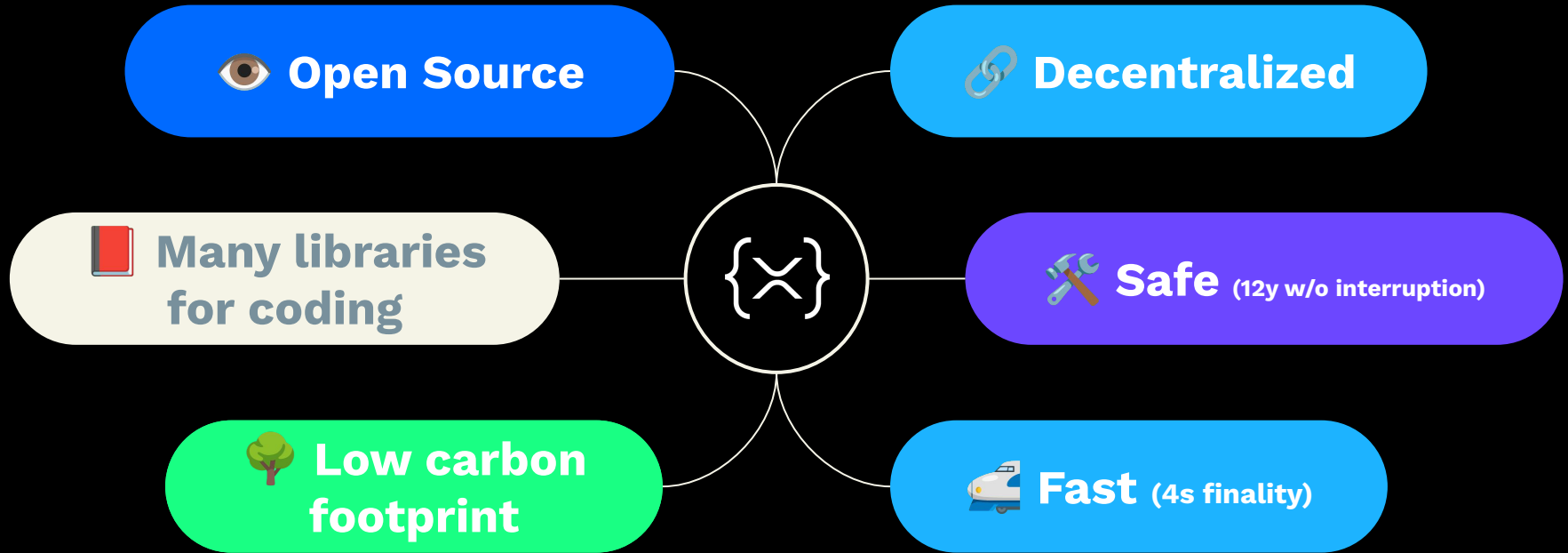
Building on the XRP Ledger Workshop



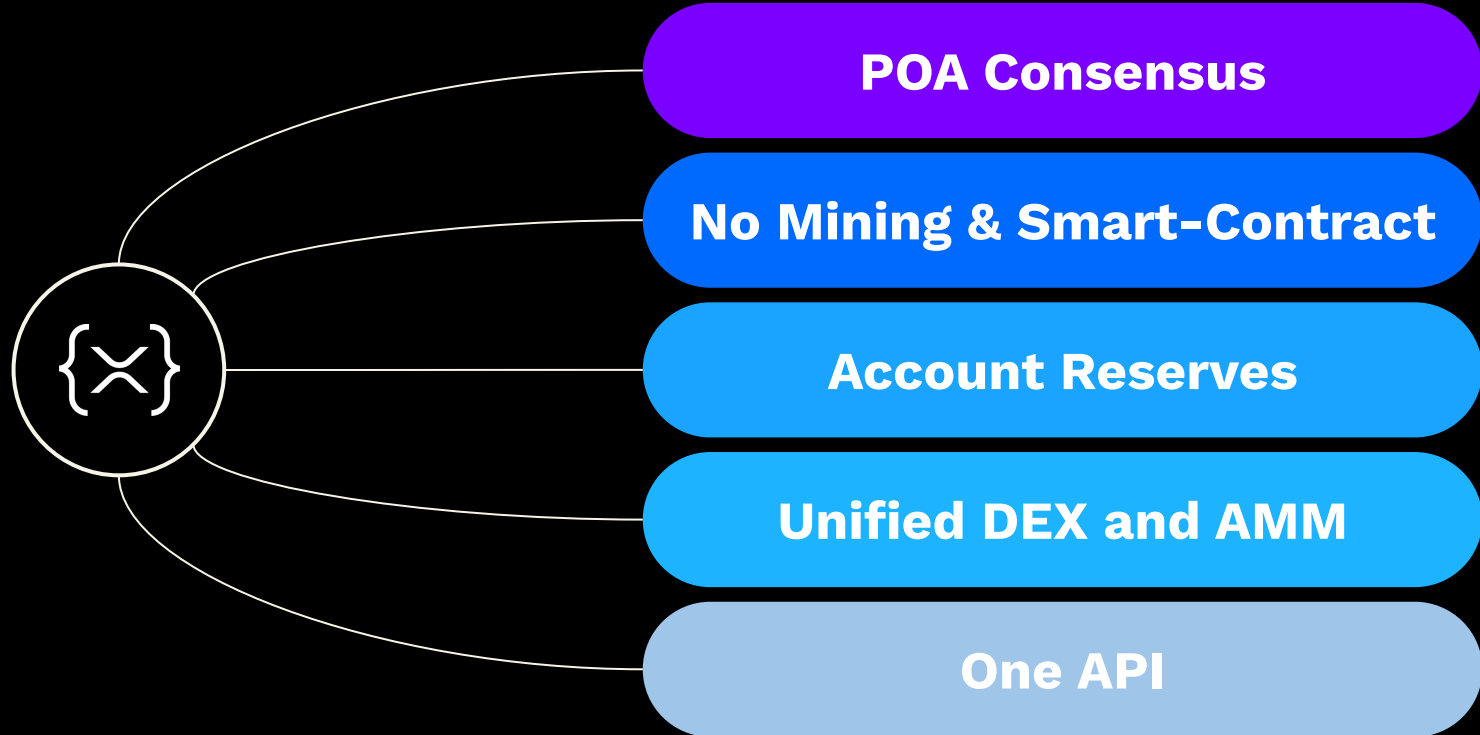
Maxime DIENGER

Developer Advocate
Ripple

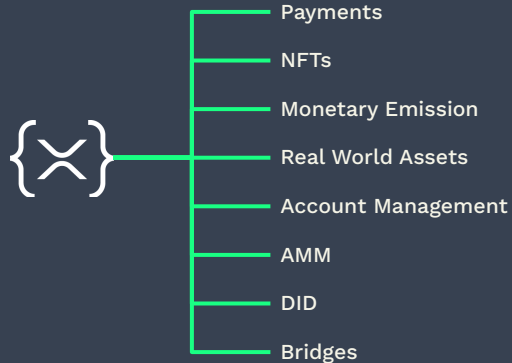
The XRP Ledger



Specificities of the XRPL



One endpoint to do all the things



Single API

No need to stitch together disparate systems or spend months integrating complex technology - simply connect into XRPL through a single API

Minimal Code Required

Astonishingly simple, you can get up and running on the XRP Ledger in as little as few lines of code using familiar programming languages (JS, Python, Java, and many more)

Powerful protocol features provide the building blocks to innovate

Native DEX

First on-chain DEX in the world, trading and moving tokens anywhere in seconds with competitive liquidity

Issued Assets

Ability to represent digital currencies, legal obligations, fungible tokens, and other asset classes on the ledger

Non-Fungible Tokens

Implements non-fungible tokens with built-in royalties where all trades handled by the DEX

Token Asset Controls

Controls for token issuers and holders to enhance security and regulatory compliance

Advanced Payments

Use advanced payment capabilities like “Escrow” and “Checks” to build smart applications without smart contracts

Automated Market Maker

Liquidity pools bring yielding assets to the ledger as well as the ability to provide liquidity on your tokens

Protocol contributors are building a full suite of institutional-grade capabilities to expand use cases on XRPL

● Amendment up for voting

Batch Tx

Allows multiple transactions to be bundled into a batch that's processed all together.

● Live on XRPL Mainnet !

MPTokens

Implements a new type of fungible token, optimized to be used for common token use cases.

● Amendment up for voting

TokenEscrow

Extends the existing Escrow functionality to support escrowing issued tokens (IOU) or MPTs.

● In Development

Smart Escrow

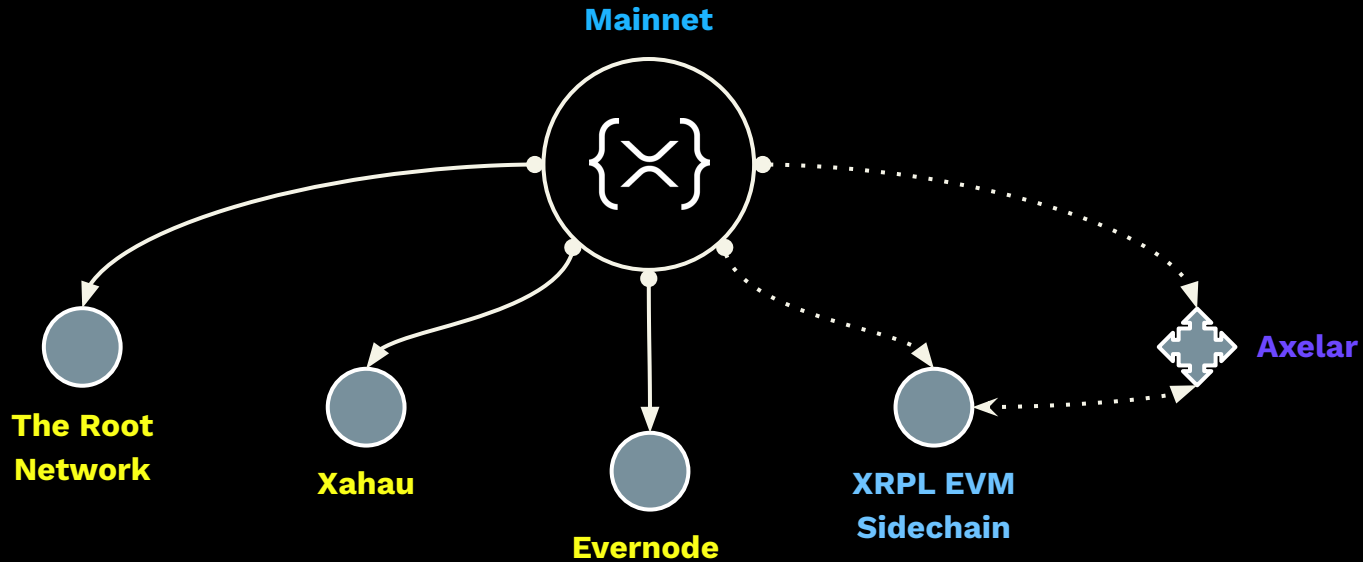
Programmable escrow that uses custom WASM code to enforce flexible, on-chain conditions beyond simple time locks or hashlocks.

● In Development

Lend / Borrow

Protocol native lending and borrowing functionality without needing smart contracts.

The XRPL Extended Ecosystem



XRPL Mainnet interoperates with Axelar, to bridge XRP and tokens



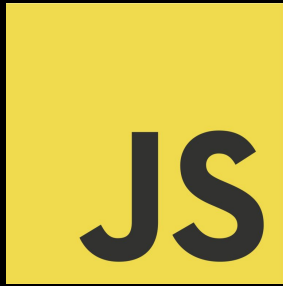
Code examples

XRPL - Client Libraries

Recommended: JavaScript/TypeScript (xrpl.js) or Python (xrpl-py)

What is a client library ?

A client library is a language-specific package that provides prebuilt methods and typed models to simplify interacting with an API.



General Process to Interact with the XRPL

01

Connect to a node

02

Create or import a wallet

03

Prepare the transaction

04

Get the result

1. Connect to a Node

```
import { Client } from "xrpl";

const client = new Client("wss://s.altnet.ripple.test.net:51233");

async function main() {
  await client.connect();
  await client.disconnect();
}
```

2. Create a Wallet

```
// Generate key pair and call the faucet
async function createWallet(client: xrpl.Client) {
  const { wallet, balance } = await client.fundWallet();
}
```

3. Prepare & Send the Transaction

```
async function sendPaymentTx(  
  client: xrpl.Client,  
  wallet: xrpl.Wallet,  
  address: string,  
  amount: number,  
) {  
  const tx: xrpl.Payment = {  
    TransactionType: "Payment",  
    Account: wallet.classicAddress,  
    Destination: address,  
    Amount: xrpl.xrpToDrops(amount),  
  };  
  
  return await client.submitAndWait(tx, {  
    autofill: true,  
    wallet,  
  });  
}
```

4. Get the Result

```
api_version: 2,
id: 22,
result: {
  close_time_iso: "2024-12-06T13:39:10Z",
  ctid: "C02C0E9E00010001",
  hash: "A2D8910A45D19A91755F3B8C1E29F5FC97C438B0E117E3FBD042DAD1C9A94B98",
  ledger_hash: "8DAABD0B422F691F3E806EF0E78B8D3F5D7F7D831E97661754A6E31B5E7CEA7C",
  ledger_index: 2887326,
  meta: {
    AffectedNodes: [ { ModifiedNode: [Object] }, { CreatedNode: [Object] } ],
    TransactionIndex: 1,
    TransactionResult: "tesSUCCESS",
    delivered_amount: "10000000"
  },
  tx_json: {
    Account: "rELCjuVzgjBrBUexXoytR59gzEMTU3BwBc",
    DeliverMax: "10000000",
    Destination: "rf26gfMAfxaSK8cRJ8b3HpSn11N4v5xD9h",
    Fee: "12",
    Flags: 0,
    LastLedgerSequence: 2887344,
    Sequence: 2887324,
    SigningPubKey: "ED598042DFC6F9C65B16002F042B57AA4372EC6B12B9F3862F772A4FCF7B331B8C",
    TransactionType: "Payment",
    TxnSignature: "4ABCFD76572234A1B51BE6509C6364835530EE44C2B59005D4A5ACE4C84B13F9EF0025CF84A5C1F922F500877E2BCFA
0C6696A434024FD2E3A776E0F7B485C07",
    date: 786807550,
    ledger_index: 2887326
  },
  validated: true
},
type: "response"
```

The entire code

```
import xrpl from "xrpl";

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

async function main() {
  // 1. we connect to a node
  await client.connect();

  // 2. we create a wallet
  const wallet = await createWallet(client);

  // 3. we prepare and send the tx
  // Here we want to send 10 XRP to rf26gfMAfxaSK8cRJ8b3HpSn11N4v5xD9h
  const amount = 10;
  const tx = await sendPaymentTx(
    client,
    wallet,
    "rf26gfMAfxaSK8cRJ8b3HpSn11N4v5xD9h",
    amount,
  );

  //4. we get the result
  console.log(tx);
  await client.disconnect();
}
```

Tokens on XRPL - trustlines

A promise between parties

- An Issuer creates a token representing an asset (like USD, EUR...)
- Users establish Trustlines with issuers they want to receive tokens from
- When you hold a token, you're actually holding a promise from the issuer, the same concept behind some stablecoins (RLUSD, USDC and so)

Key Components

- **Issuer:** The entity backing the token's value and responsible for redemption
- **Trustline:** A connection showing you trust an issuer up to a specific amount
- **Token:** The digital IOU that can be transferred between users who trust the same issuer

Real-World Example:

When you hold **RLUSD** on XRPL or Ethereum, you're holding **Ripple's promise to redeem 1 USD for each token.**

You can only receive these tokens after establishing **trust with Ripple as an issuer.**

Interacting with trustlines



Set up Trust

- Tell the network that you trust an issuer

Get Tokens

- Buy from the Dex or the AMM
- Get some from the issuer

Use your Tokens

- Send to others
- Remove trust when done
- Provide liquidity

Code: Interacting with trustlines

```
const trustSet: TrustSet = {
  TransactionType: 'TrustSet',
  Account: receiver.address,
  LimitAmount: {
    currency: tokenCode,
    issuer: issuer.address,
    value: '5000000000' // 500M tokens
  },
  Flags: TrustSetFlags.tfClearNoRipple
}
console.log(trustSet)

// Receiver opening trust lines
const preparedTrust = await client.autofill(trustSet)
const signedTrust = receiver.sign(preparedTrust)
const resultTrust = await client.submitAndWait(signedTrust.tx_blob)

console.log(resultTrust)
console.log('Trust line issuance tx result: ', resultTrust.result.hash)
```

How to get RLUSD on Testnet ?

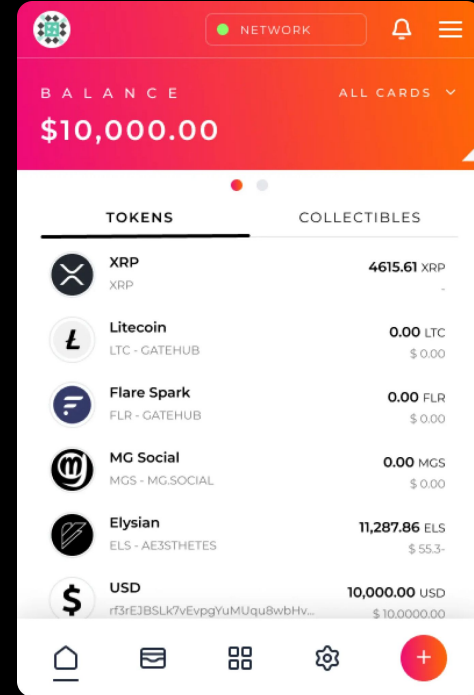


tryrlusd.com

RLUSD on XRPL Testnet: use Crossmark



crossmark.io



XRPL Amendment Testing Suite



<https://tests.xrpl-commons.org/>

xrpl-connect



<https://github.com/XRPL-Commons/xrpl-connect>

Scaffold-XRP



<https://github.com/XRPL-Commons/scaffold-xrp>

Bedrock



<https://github.com/XRPL-Commons/Bedrock>

Resources



<https://linktr.ee/rippledevrel>

