BORGIASYSTEMS LLC

+1 510-513-3222
www.borgiasystems.com
contact@borgiasystems.com
42808 Christy St, Ste 122,
Fremont, CA 94538

Interactive Hooks

Mission:

The Interactive Hooks with Smart Contract Application mission is to create a simple solution by providing

1. a user-friendly UI to the end user to create a Smart Contract on the XRPL (Hook) and execute the Smart Contract without the necessity of writing in C
2. a REST API to developer community to create Smart Contract on the XRPL (Hook) and execute the Smart Contract without the necessity of writing in C

Goal:

The goal of the Interactive Hooks Application project is to increase adoption of Hooks by making the deployment of smart contracts on the XRPL more intuitive and accessible for developers and end users of the XRPL Community.

Business Objective:

The objective of Interactive Hooks Application is to simplify the process of exchanging asset ownership between parties with the help of Smart Contract using the XRPL .

+1 510-513-3222
www.borgiasystems.com
contact@borgiasystems.com
42808 Christy St, Ste 122,
Fremont, CA 94538

Problem and Solution:

Hooks can add an incredible amount of utility to the XRP Ledger, but in the current stage they are largely inaccessible not only to the general XRPL retail holder, but to a large majority of the XRPL developer community as well. Implementing Hooks requires a special skillset.

The main reason for this is that XRPL developers are required to interact with Hooks using the C programming language, which is relatively more complex and difficult to work with when compared to Java, JavaScript, or its derivative languages such as js.node.

Solution:

Interactive Hooks Application proposes to increase adoption and ease of use of the XRPL's Hooks smart contract functionality by creating a thin translation layer, whereby end users and developers who are not conversant in C programming languages can implement Hooks with smart contracts for their wallets or apps using an intuitive, idiot-proof interface and/or light weight REST API Interaction.

Use Case Examples:

Example 1:

Implement smart contracts which enable automated charitable donations for XRPL users or developers. For example, imagine a user who wants to donate 10 XRP for every 1000 XRP they send or spend.

Currently, they would have to do this manually, but with our proposal, they could use our interface to create a simple Hooks smart contract involving the two wallets (sender and receiver) but a third wallet belonging to the registered nonprofit to which the tax-deductible donation could be sent.

Fellow XRPL Grants recipient the **Center for Collaborative Economics** have offered their platform for an initial use case for this automated donation functionality

Example 2:

If our user desires to buy or sell an asset, to avoid the involvement of conciliator to exchange the documents and payment transactions, it's a tedious task for both participants in their daily lives, and it is a labour-intensive process to exchange the ownership of the asset and to guarantee Fraud Prevention.

To overcome the complications with the traditional process of exchanging ownership of the assets with terms and conditions , Interactive Hooks Application is the solution to execute without conciliator involvement. This application enables the utilization of Hooks without writing C program to automate the enforcement of terms faster than traditional contracts. Whenever the Hook conditions are met, the Smart Contract executes and money will be transferred automatically to the party’s account.

Process of implementing Interactive Hooks with Smart Contract:

Interactive Hooks with Smart Contract Application will collect the required information from the end user as per the requirement from the provided UI, with the provided information by the user Interactive Hooks application will create a hook. Once the hook is created then our application will fetch the data from the UI and will create a WASM file with set of instructions that need to be executed.

To proceed further, the set of instructions will be shared by the end user which will assign the hook as per the requirement and will share the hook details to the XRPL network. Once the user has initiated the Smart Contract, the assigned hook will validate the instructions provided by the end user. Once all the instructions provided by the end user are valid the contract will be executed as specified by the user .

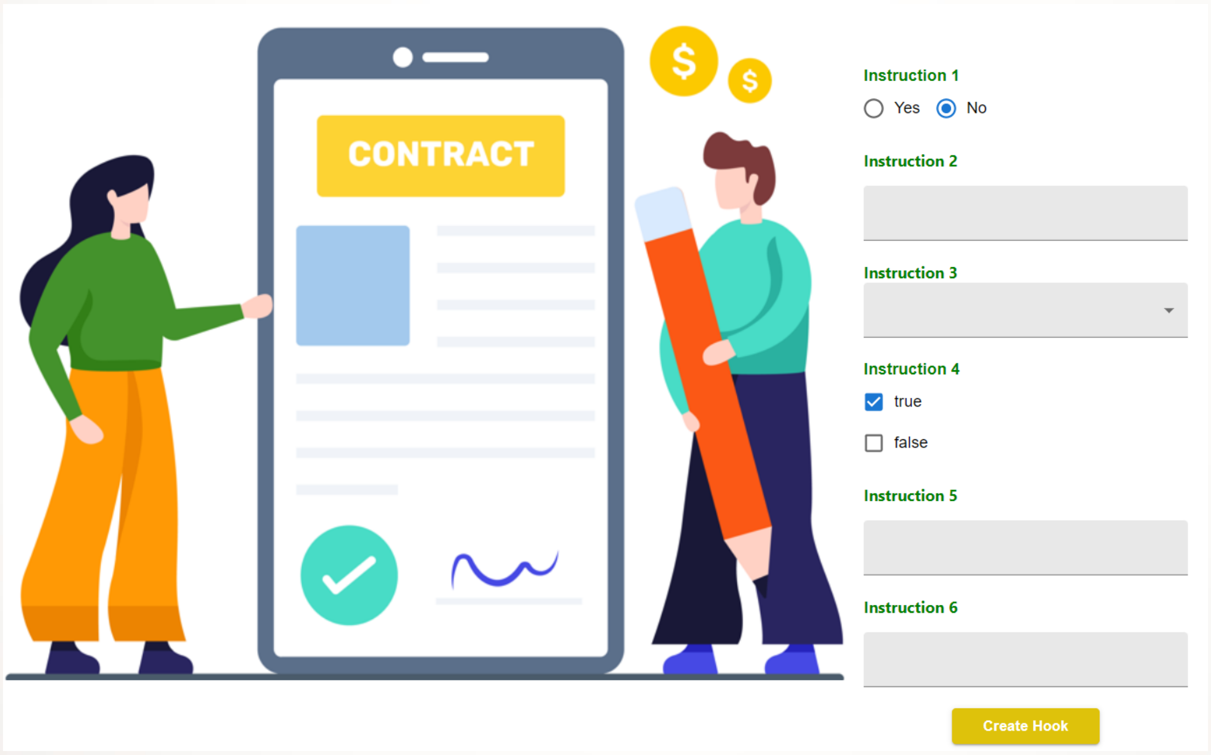
How Interactive Hooks is Developer and End User friendly:

- In Interactive Hooks, the end user is interacting with our UI which is a user-friendly interface to provide all the required instructions as per the requirement
- Interactive Hooks builds on Opensource licensing with **creativecommons.org**, hence developers can integrate and enhance easily and openly
- End users won't need to possess technical expertise to create a hook or execute smart contracts using the Interactive Hooks Application
- End users are not required to login to their XRPL account to perform any transaction executing over Interactive Hooks Application.

Functional Design:

End User UI Flow:

1. Interactive Hooks with Smart Contract Application collects all the instructions form the end user.



Instruction 1

Yes [✓ No]

Instruction 2

Instruction 3

Instruction 4

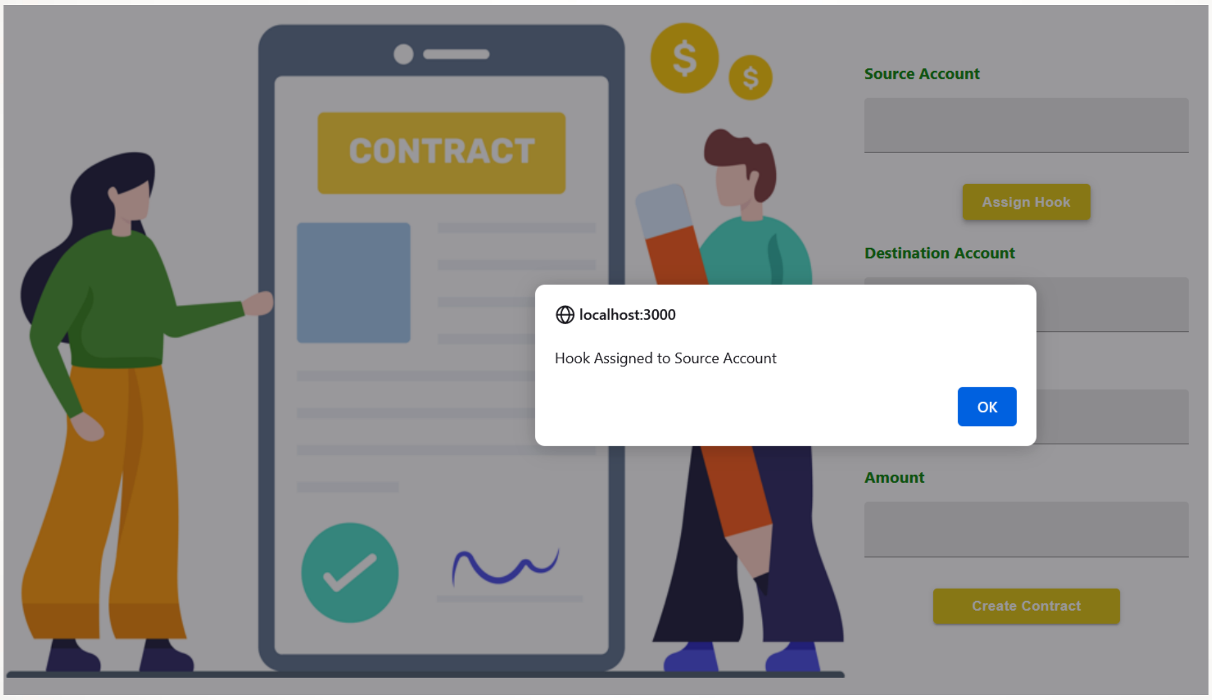
[✓ true]

false

Instruction 5

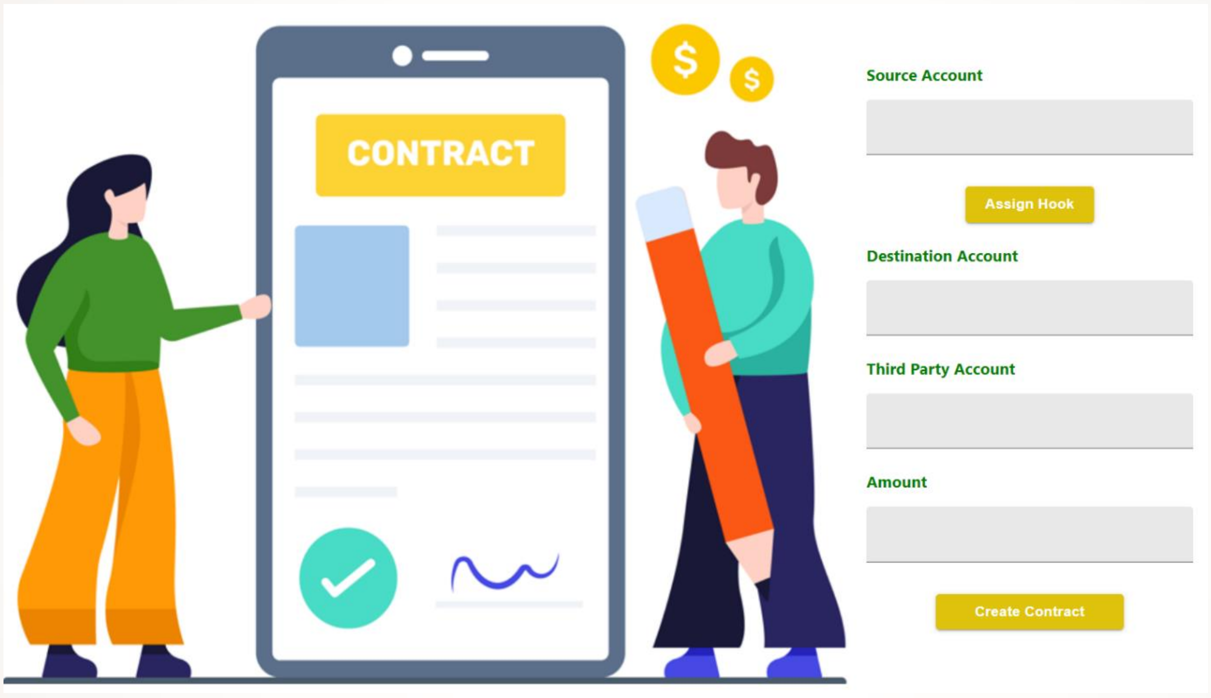
Instruction 6

2. When end user click on Create Hook button Interactive Hooks with Smart Contract Application will create Hook with provided instructions by the end user.

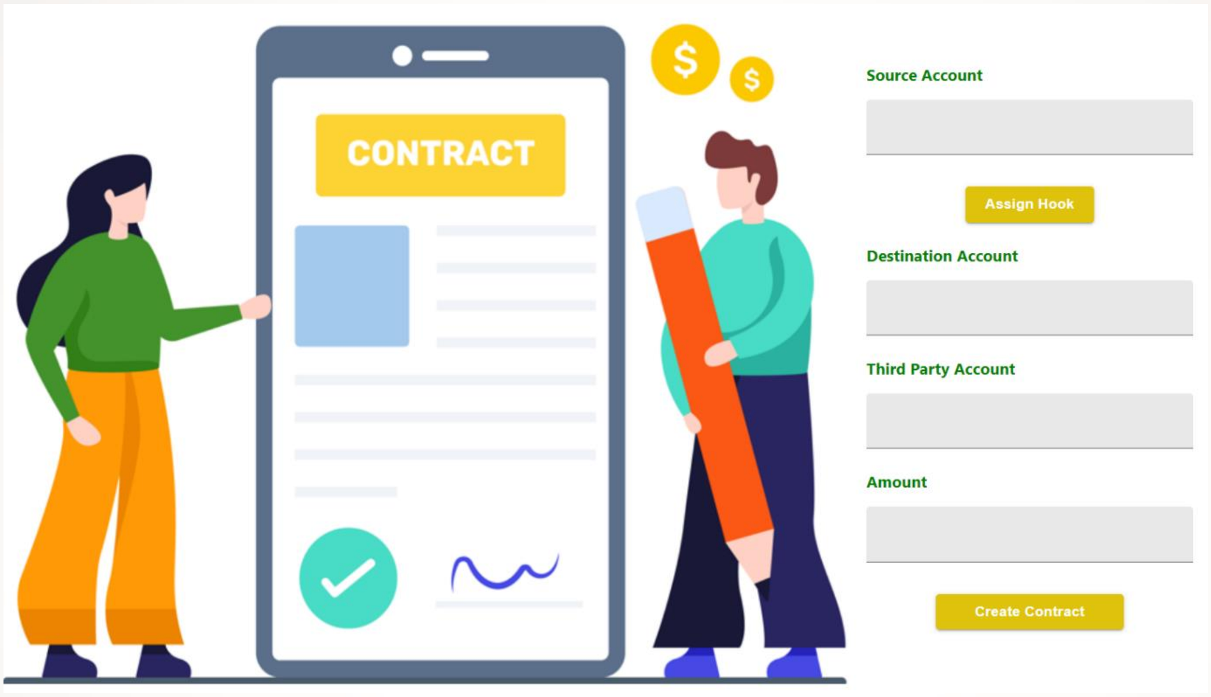


Hook Assigned to Source Account

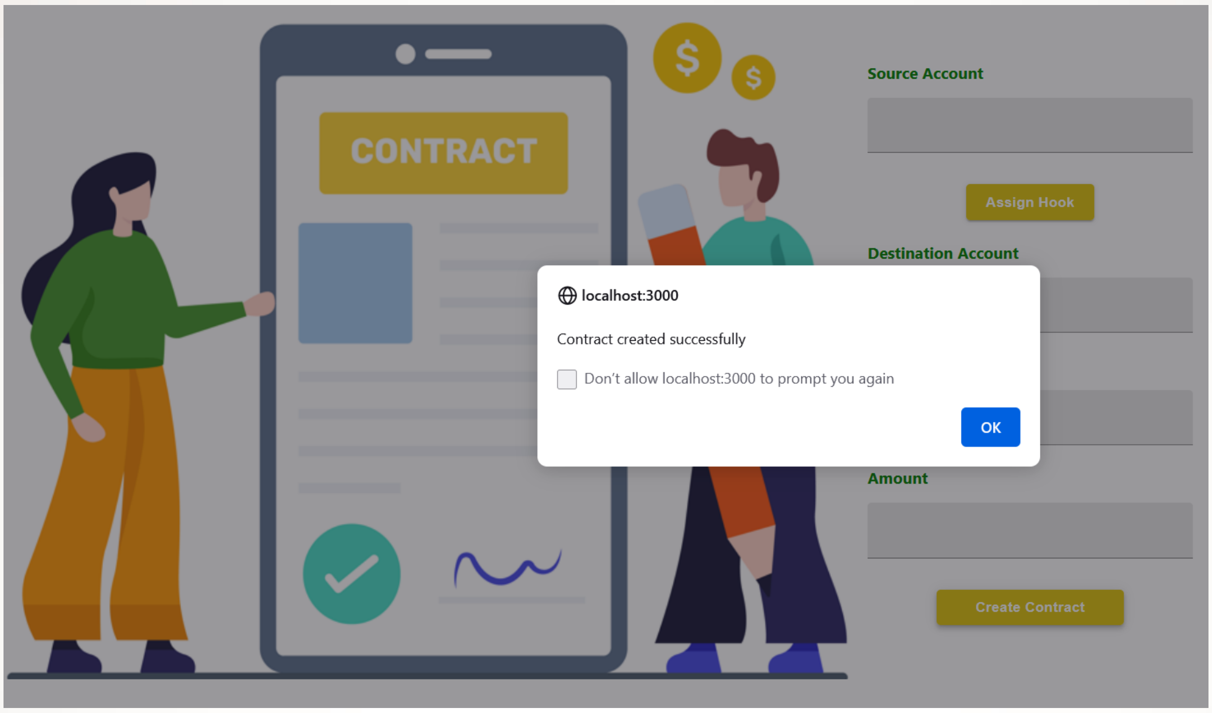
3. In Next Screen Interactive Hooks with Smart Contract Application collects account details to assign hook whenever user clicks on Assign Hook button.



4. After assigning hook user enters transaction details to create smart contract using Create Contract button.



5. Once contract execution is successful, User receives Success response through alert box.


localhost:3000

Contract created successfully

Don't allow localhost:3000 to prompt you again

6. For any transaction execution failures , user receive failure error from XRPL and display in UI.

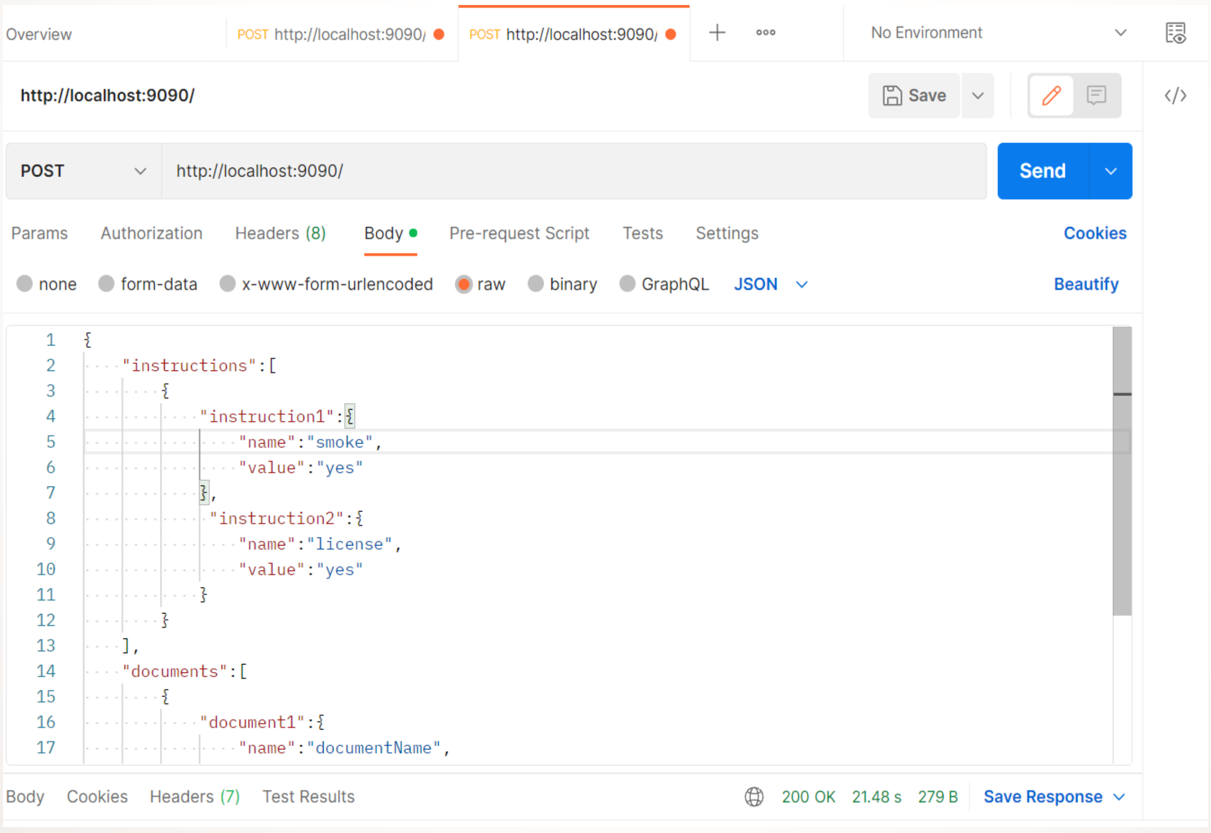
JavaScript based REST API Integration Functional Overview:

createHook:

1. End point URL to test: http://localhost:9090/
2. Request type: POST
3. API request collect instructions consume createHook operation
4. Sample request payload:

```
{
   "instructions”: [
      {
         "instruction1":{
            "name":"Test Instruction 1",
            "value":"Sample text"
         },
          "instruction2":{
            "name":" Test Instruction 2",
            "value":"yes"
         }
      }
   ],

}
```

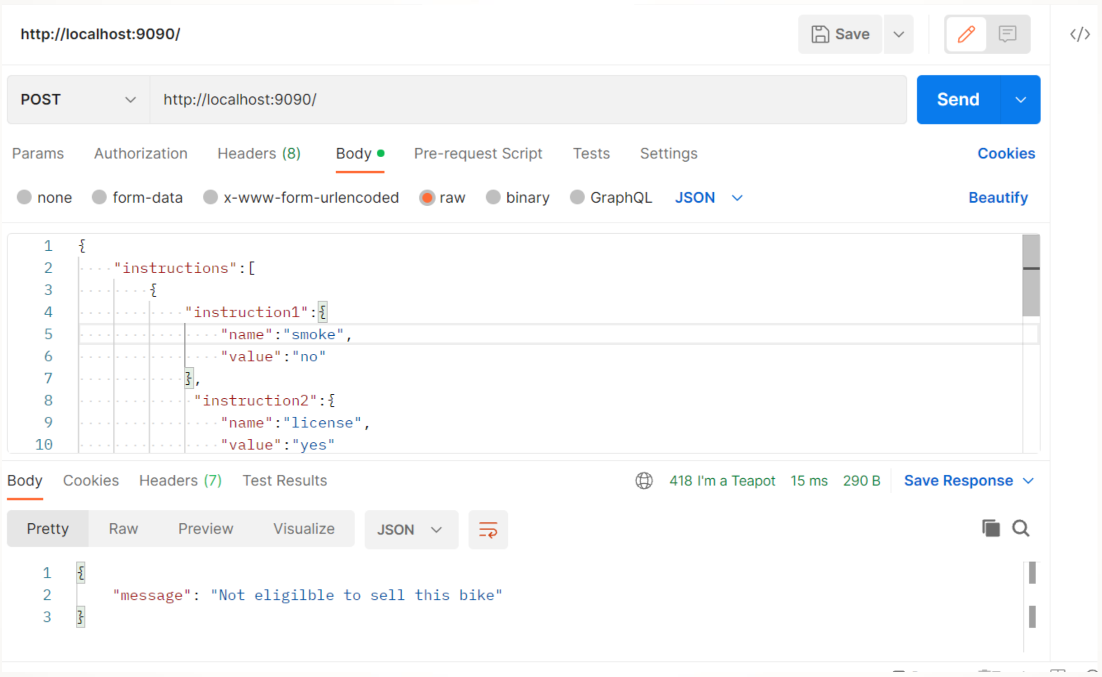
5. REST Service
   - Select POST request type.
   - Select raw as request format type and then select JSON format.
   - Provide JSON request in proper format.
   - Click on Send button to submit request to InteractiveHooksAPI.



POST http://localhost:9090/

```
{
    "instructions":[
        {
            "instruction1":{
                "name":"smoke",
                "value":"yes"
            },
             "instruction2":{
                "name":"license",
                "value":"yes"
            }
        }
    ],
    "documents":[
        {
            "document1":{
                "name":"documentName",
```

6. System creates a hook and pass with response with hook details.



POST http://localhost:9090/

```
{
    "instructions":[
        {
            "instruction1":{
                "name":"smoke",
                "value":"no"
            },
            "instruction2":{
                "name":"license",
                "value":"yes"
```

418 I'm a Teapot 15 ms 290 B

```
{
    "message": "Not eligilble to sell this bike"
}
```

executeSmartContractwithHook:

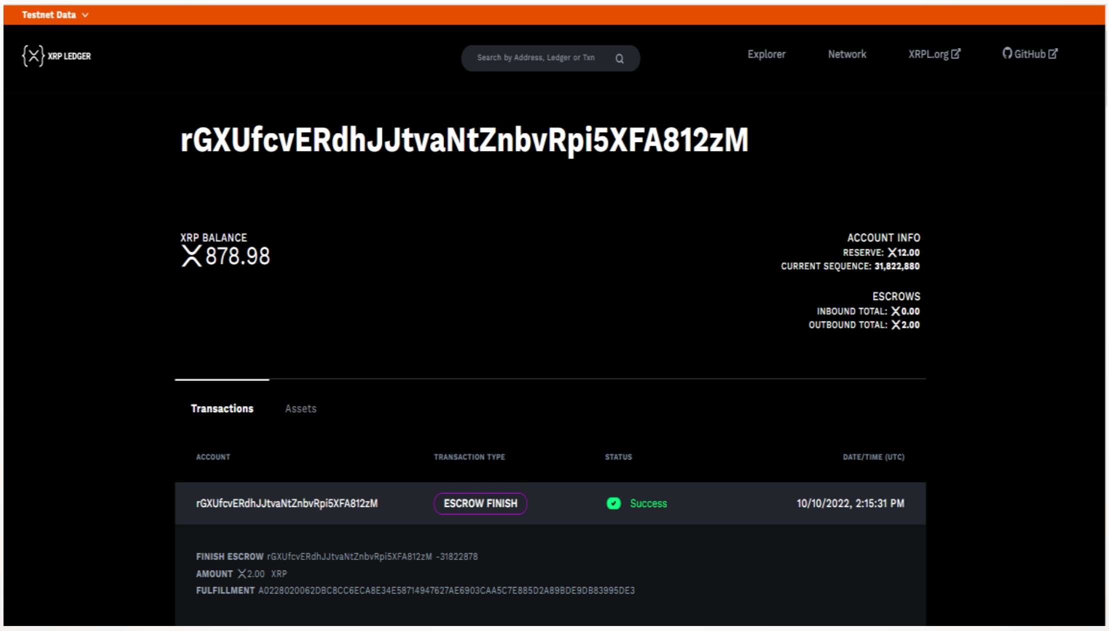
1. End point URL to test: http://localhost:9090/
2. Request type: POST
3. API request collect instructions consume **executeSmartContractwithHook** operation
4. Sample request payload:

```
{
   "SmartContractDetails”:
      {
         "HookID":{

            "value":"Sample text"
         },
          "SenderAccount":{
            "AccountAddress":" TestAccount",
            "Amount":"1000"
         }
          "ReceivedAccount":{
            "AccountAddress":" TestReceiverAccount",
         }
          "ThirdPartyAccount":{
            "AccountAddress":" TestTPAccount",
         }

      }
   ,

}
```

5. REST Service
    - Select POST request type
    - Select raw as request format type and then select JSON format
    - Provide JSON request in proper format
    - Click on Send button to submit request to InteractiveHooksAPI
6. System assigns hooks execute smart contract and pass with response with hook details.

7. After successful transaction verify buyer and seller test accounts using the link https://testnet.xrpl.org/

8. After Smart Contract execution results can see in XRPL network



rGXUfcvERdhJJtvaNtZnbvRpi5XFA812zM

XRP BALANCE
878.98

ACCOUNT INFO
RESERVE: 12.00
CURRENT SEQUENCE: 31,822,880

ESCROWS
INBOUND TOTAL: 0.00
OUTBOUND TOTAL: 2.00

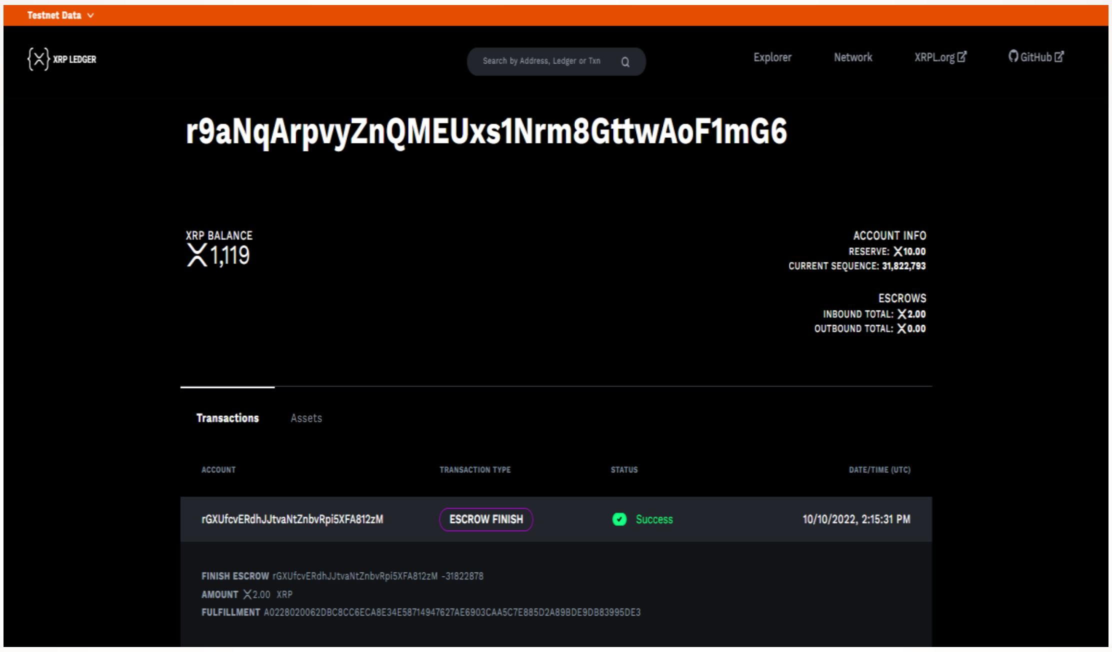
Transactions Assets

ACCOUNT	TRANSACTION TYPE	STATUS	DATE/TIME (UTC)
rGXUfcvERdhJJtvaNtZnbvRpi5XFA812zM	ESCROW FINISH	✓ Success	10/10/2022, 2:15:31 PM

FINISH ESCROW rGXUfcvERdhJJtvaNtZnbvRpi5XFA812zM -31822878
AMOUNT 2.00 XRP
FULFILLMENT A0228020062DBC8CC6ECA8E34E58714947627AE6903CAA5C7E885D2A89BDE9DB83995DE3

9.

10. Verify the seller account using seller account address.



r9aNqArpvyZnQMEUxs1Nrm8GttwAoF1mG6

XRP BALANCE
1,119

ACCOUNT INFO
RESERVE: 10.00
CURRENT SEQUENCE: 31,822,793

ESCROWS
INBOUND TOTAL: 2.00
OUTBOUND TOTAL: 0.00

ACCOUNT	TRANSACTION TYPE	STATUS	DATE/TIME (UTC)
rGXUfcvERdhJJtvaNtZnbvRpi5XFA812zM	ESCROW FINISH	✓ Success	10/10/2022, 2:15:31 PM

FINISH ESCROW rGXUfcvERdhJJtvaNtZnbvRpi5XFA812zM -31822878
AMOUNT 2.00 XRP
FULFILLMENT A0228020062DBC8CC6ECA8E34E58714947627AE6903CAA5C7E885D2A89BDE9DB83995DE3

+1 510-513-3222
www.borgiasystems.com
contact@borgiasystems.com
42808 Christy St, Ste 122,
Fremont, CA 94538

Technical Design:

Translation Layer Flow:

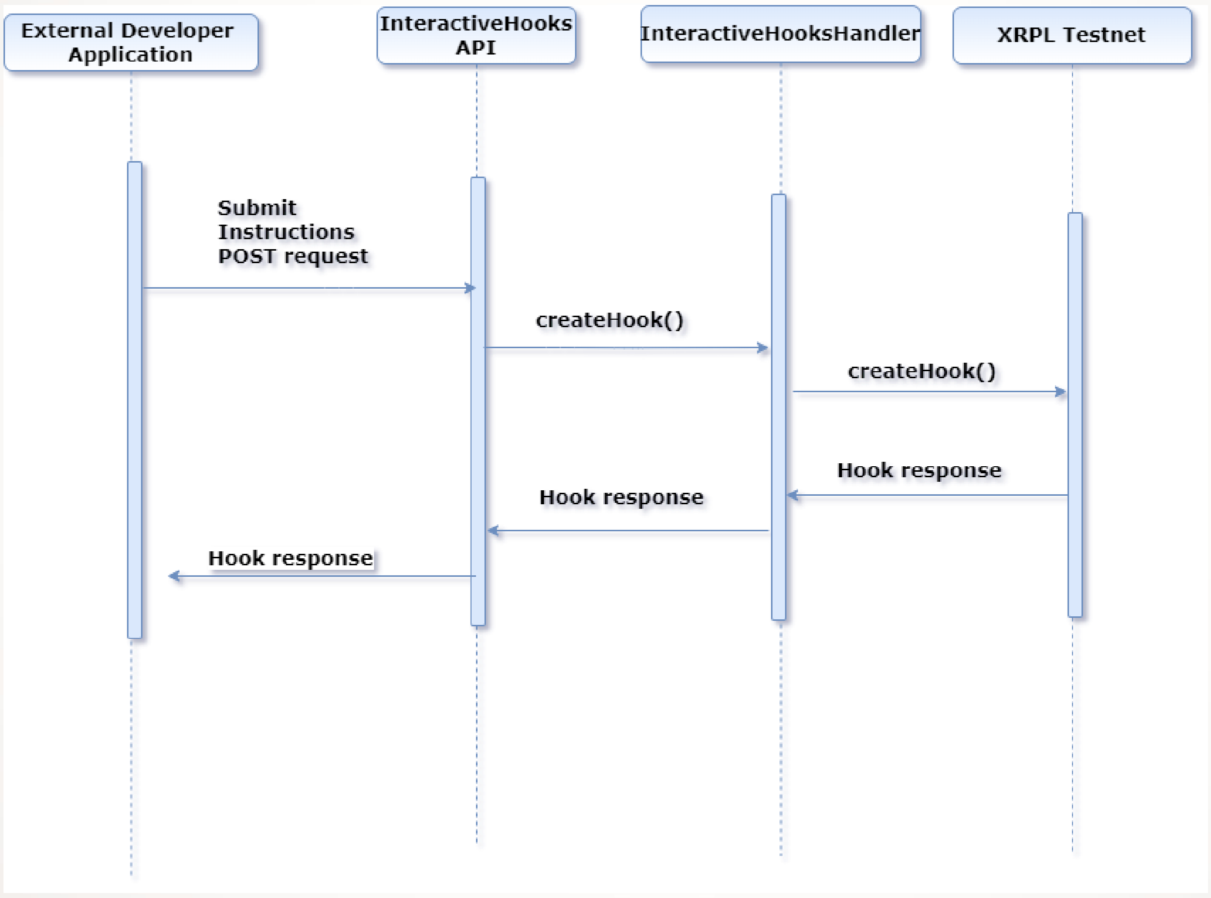
1.First create two accounts belonging to seller and buyer test accounts in the test net using this link https://xrpl.org/xrp-testnet-faucet.html and click on **Generate Testnet credentials** button to generate test accounts.

2. Configure instructions and collect instructions information from UI

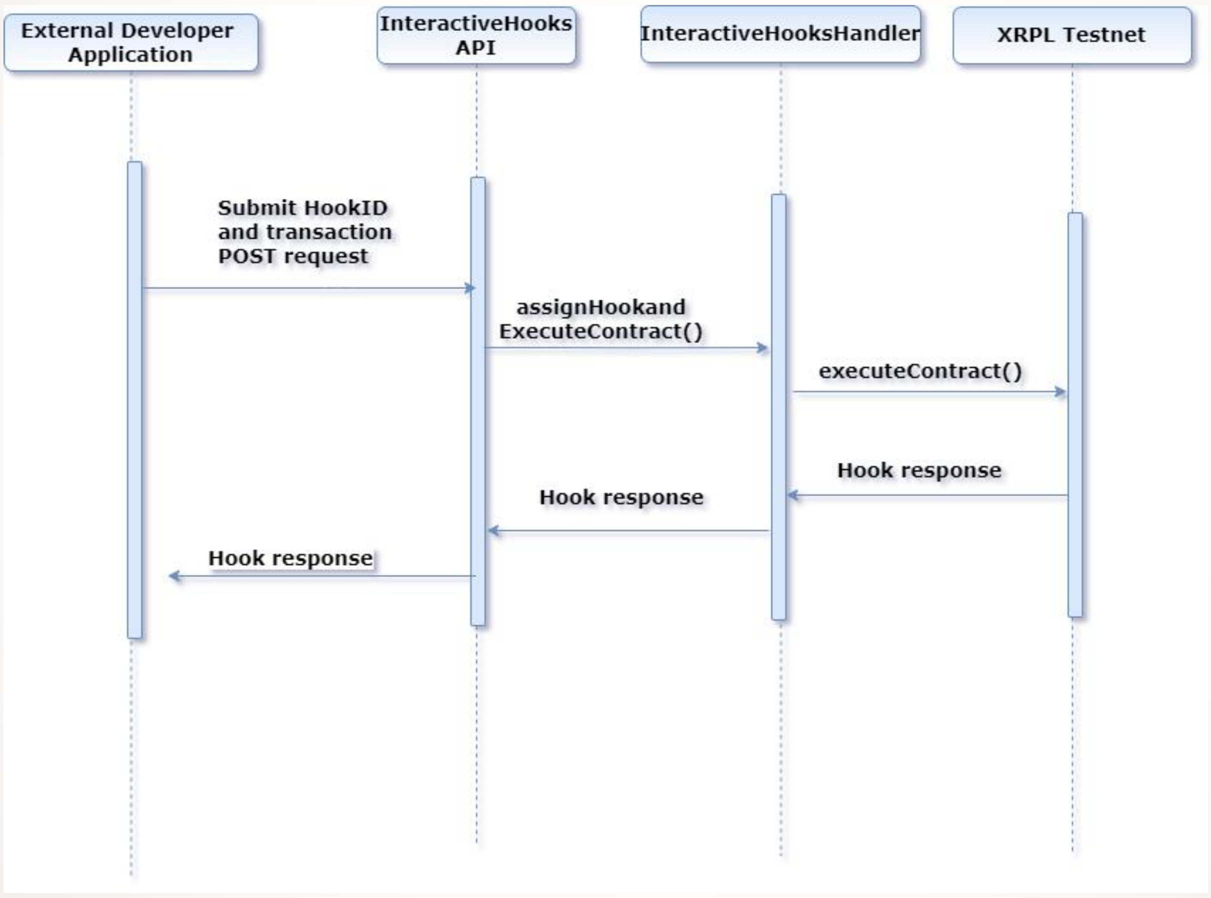
4. Convert predefined C program in to WASM
5. Populate variable data in C program with instructions data
6. Now Smart Contract Application would connect to XRPL Test net network , send the data and if this data meets the conditions the user will receive success message.
7. if this data does not meet the conditions the user will receive failure message

REST API Sequence Diagram:

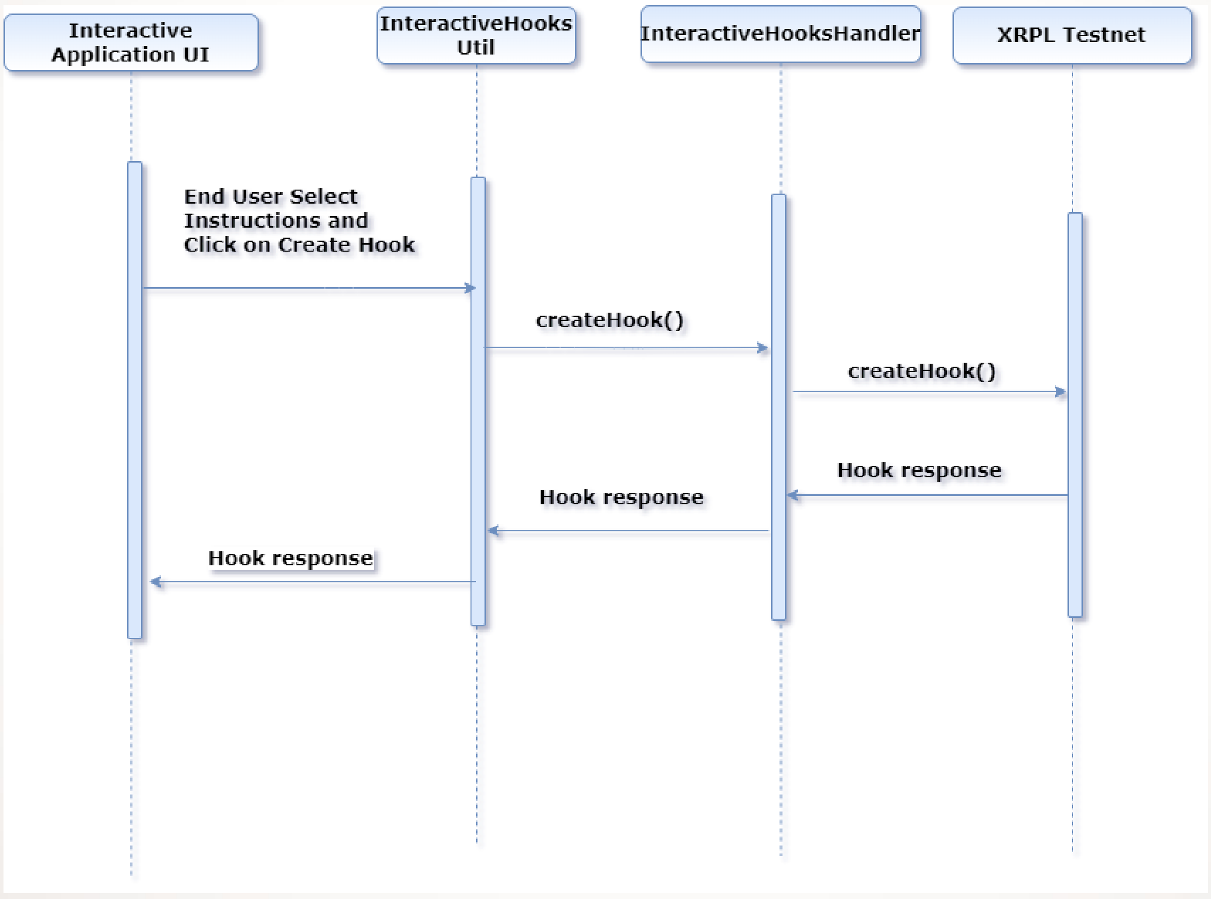
Create Hook API Sequence Diagram:



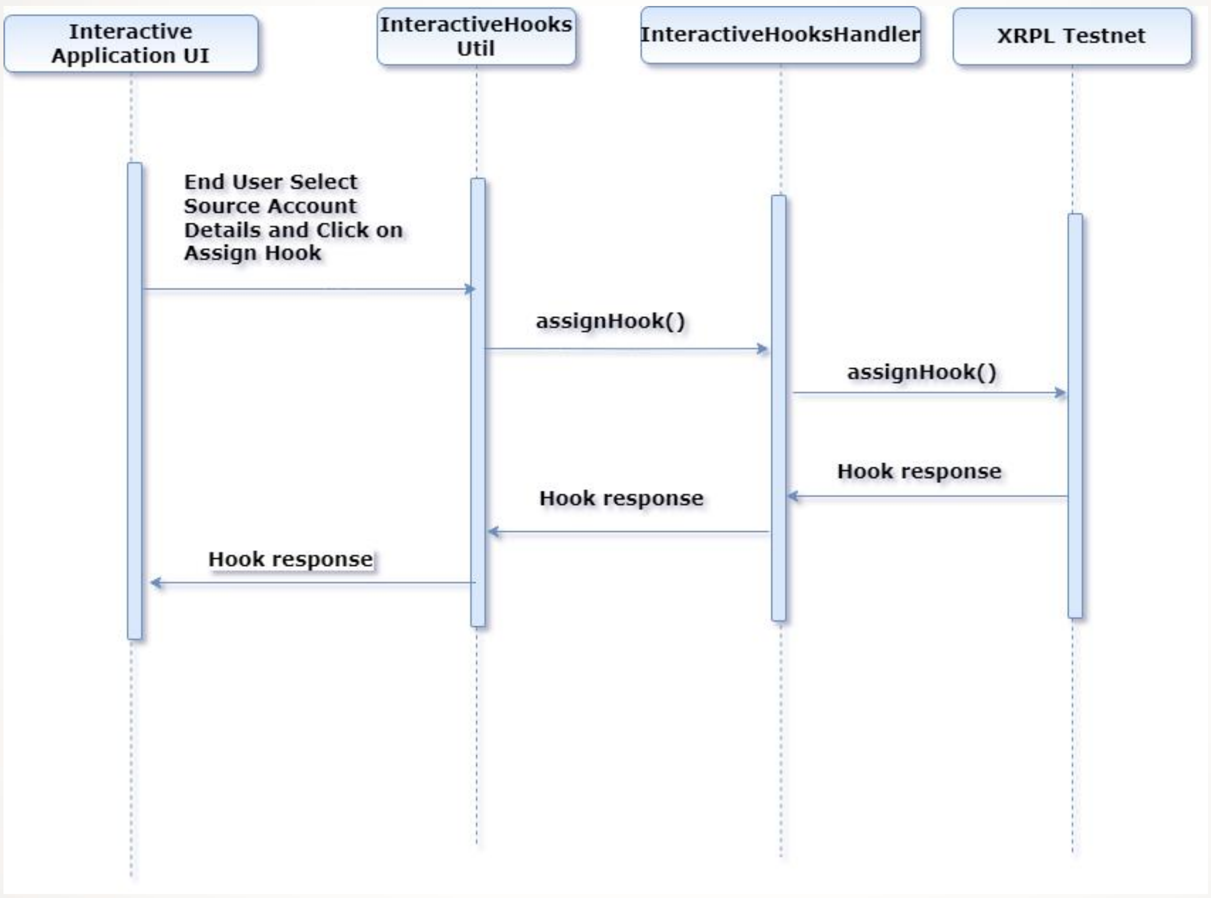
Execute Smart Contract API Sequence Diagram:



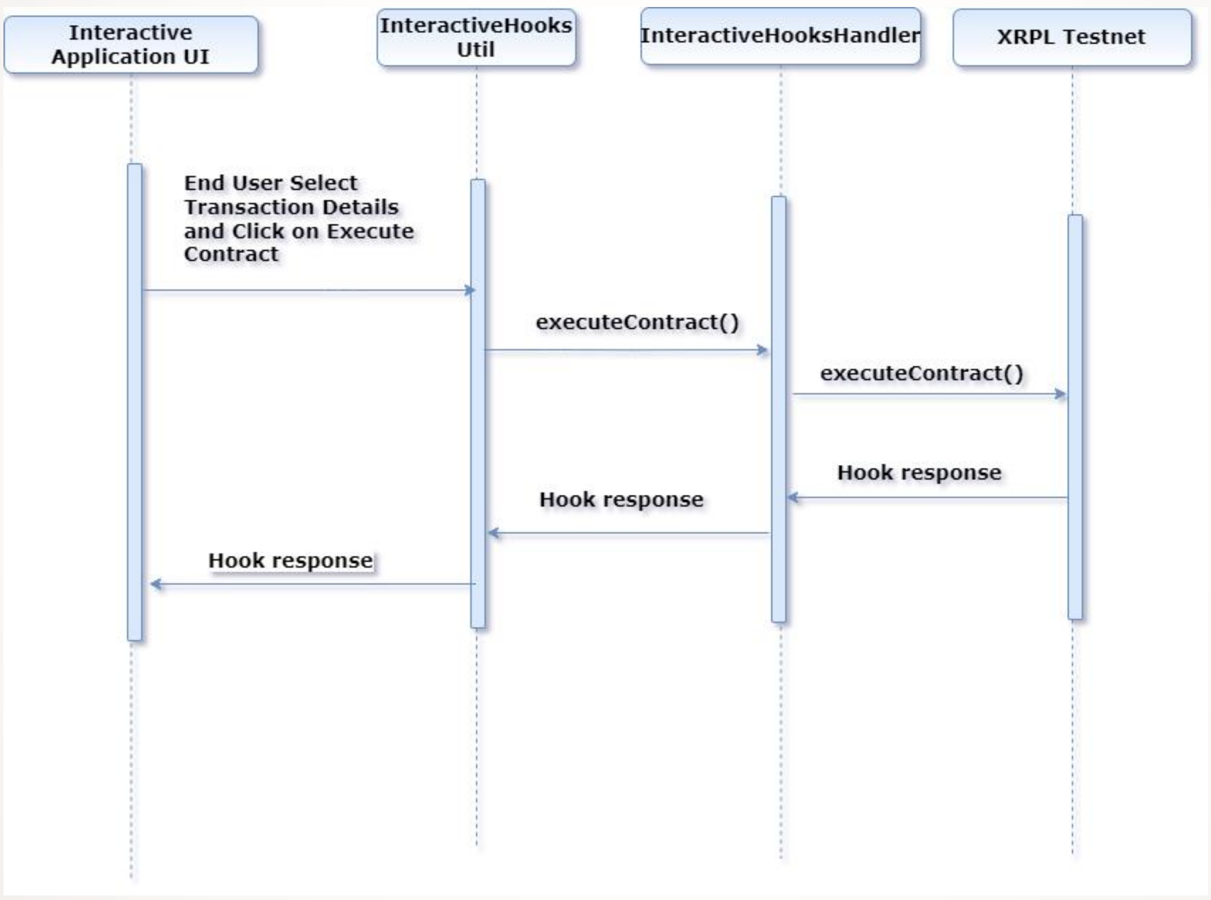
Create Hook UI Sequence Diagram:



Assign Hook UI Sequence Diagram:



Execute Smart Contract UI Sequence Diagram:



BORGIASYSTEMS LLC

+1 510-513-3222
www.borgiasystems.com
contact@borgiasystems.com
42808 Christy St, Ste 122,
Fremont, CA 94538

Integration components:

Outbound Integration:

Uses XRPL Client to create hook , assign hook and execute smart contract.

Inbound Integration:

Uses JavaScript REST API (Interactive Hooks API) to handle incoming instructions to create hook, assign hook and execute smart contracts.

Conclusion:

The deployment of the Interactive Hooks Application would represent a quantum leap forward for smart contract functionality on the XRP Ledger. The platform we propose to build could potentially lead to the minimization of indirect costs and security risks for the end user, while simultaneously making the entire XRP Ledger more efficient and functional.

Hooks stands to revolutionize the XRP Ledger by adding an entirely new level of functionality to the architecture. However, as it stands currently, this functionality is out of the hands of all but the most elite developers. We propose to democratize smart contracts on the XRP Ledger and put the power to customize and automate payments on the XRPL in the hands of the people.