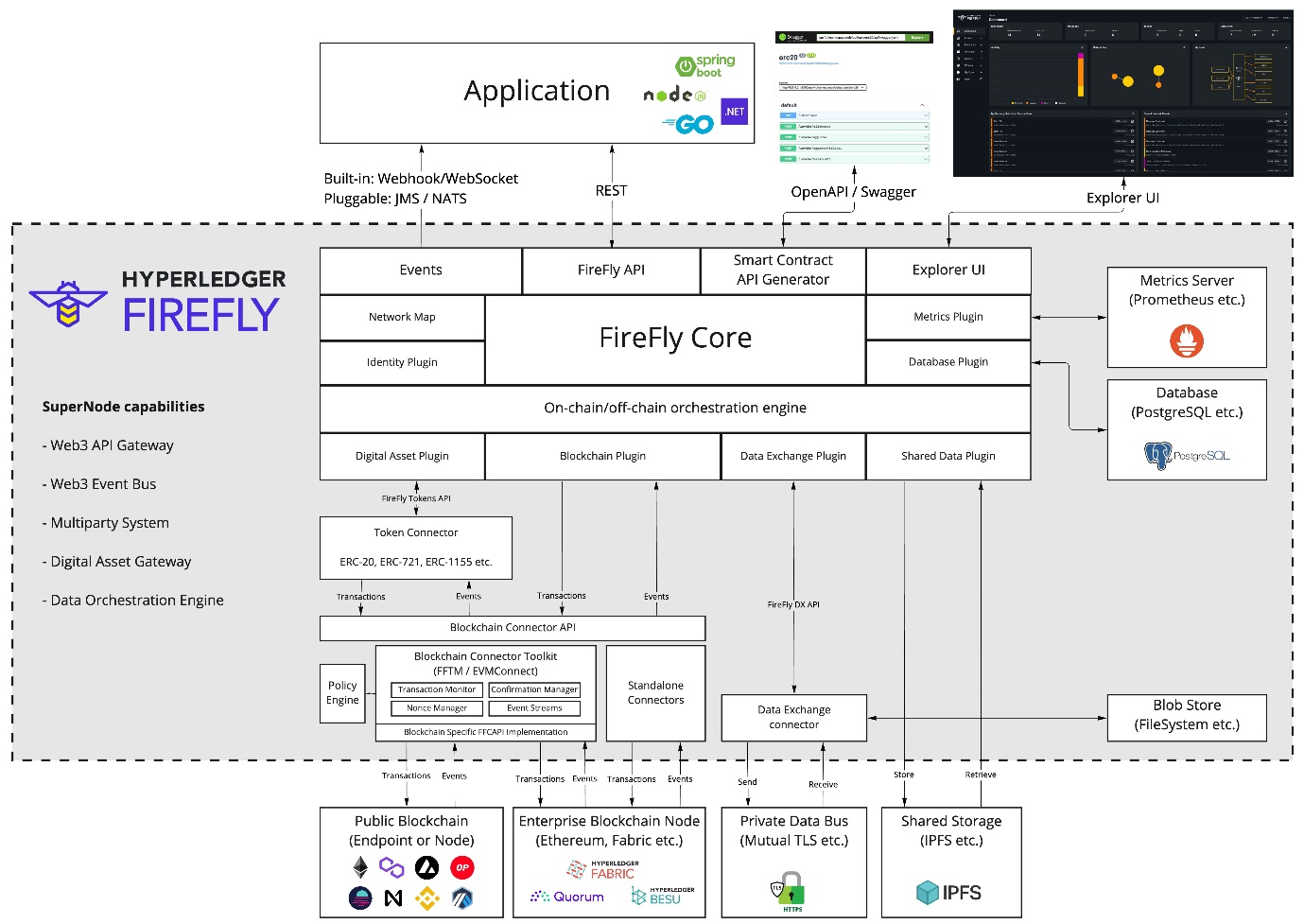
# Project Specification

Project Objective:

* To create a web application allows users to buy and sell assets.
* All the transactions are stored in block chain
* Only authorized users can view the block chain
* When an owner initiates a sale everyone in the network can view it.
* The buyer will deduct the payment after purchasing and wait for the owner to confirm the payment
* After the transaction is completed, the property owner will be updated
* The transaction can be canceled at any time during the validity period, and the transaction will be automatically closed after the validity period expires.

Developed solution:

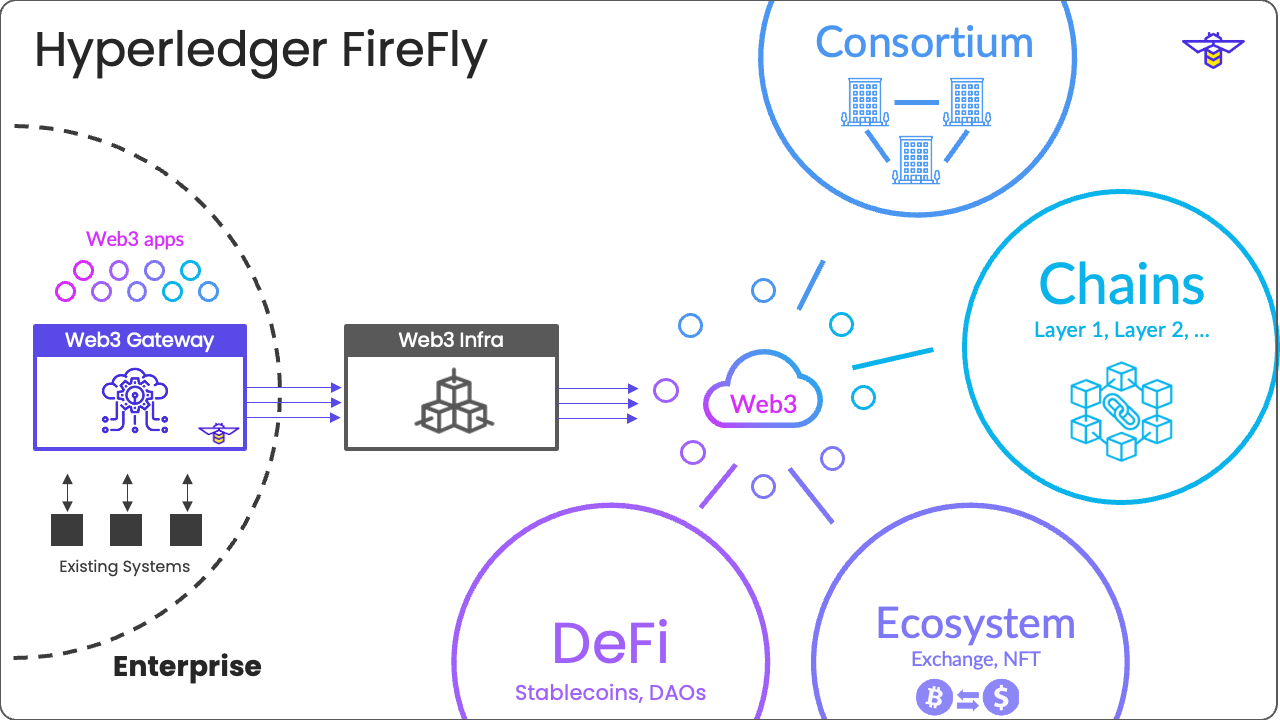
Architecture:



Technical stack used:

* Hyperledger Firefly
* Hyperledger Fabric
* fabconnect
* Firefly Nodejs SDK

## Hyperledger Firefly:



Hyperledger FireFly is an organization’s gateway to Web3, including all the blockchain ecosystems that they participate in.

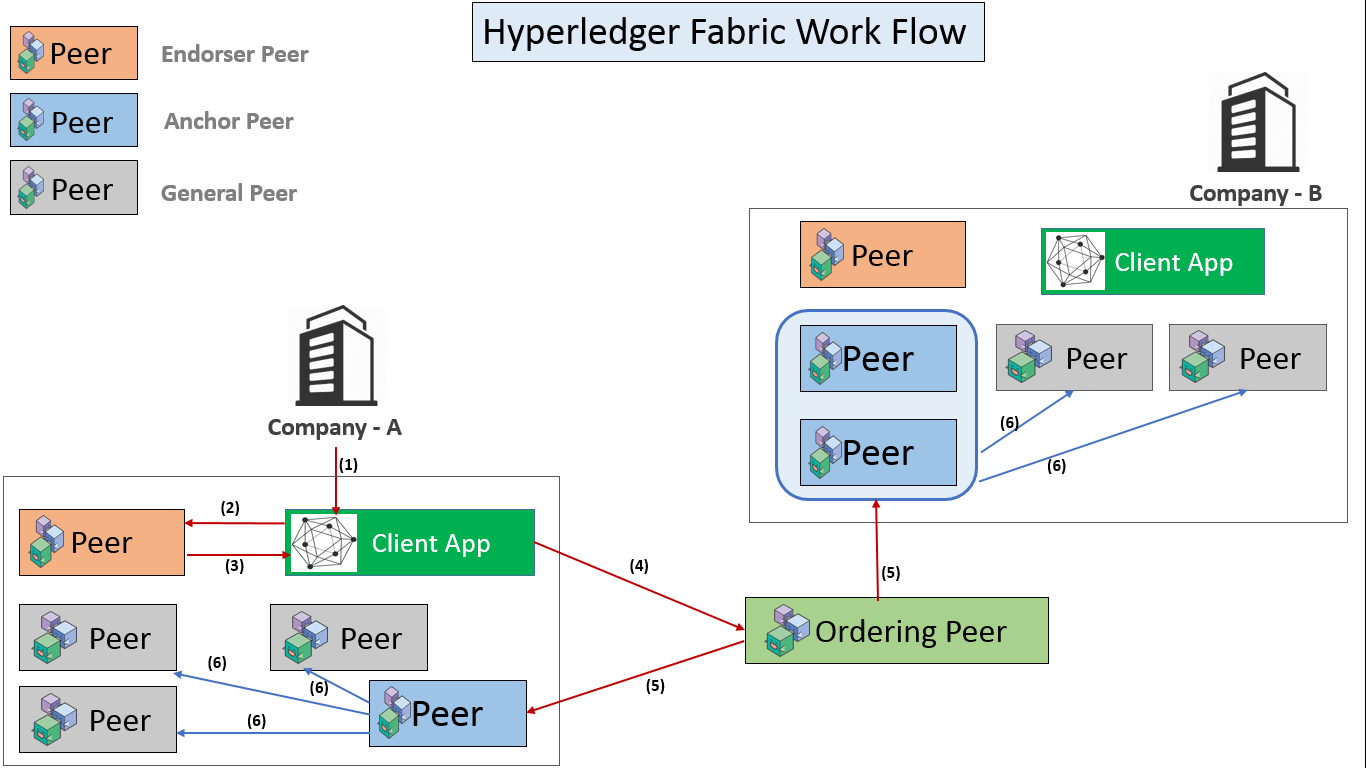
Multiple blockchains, multiple token economies, and multiple business networks.

FireFly is not another blockchain implementation, rather it is a pluggable API Orchestration and Data layer, integrating into all of the different types of decentralized technologies that exist in Web3:

* Public Blockchains, Layer 2 scaling solutions, Side chains, and App chains
* Permissioned Blockchains and Distributed Ledger Technologies (DLTs)
* Decentralized storage solutions
* Token ecosystems and standards
* Smart Contracts, DeFi solutions and DAOs
* Private off-chain encrypted communication rails
* Advanced cryptography solutions
* Identity frameworks

## Hyperledger Fabric:

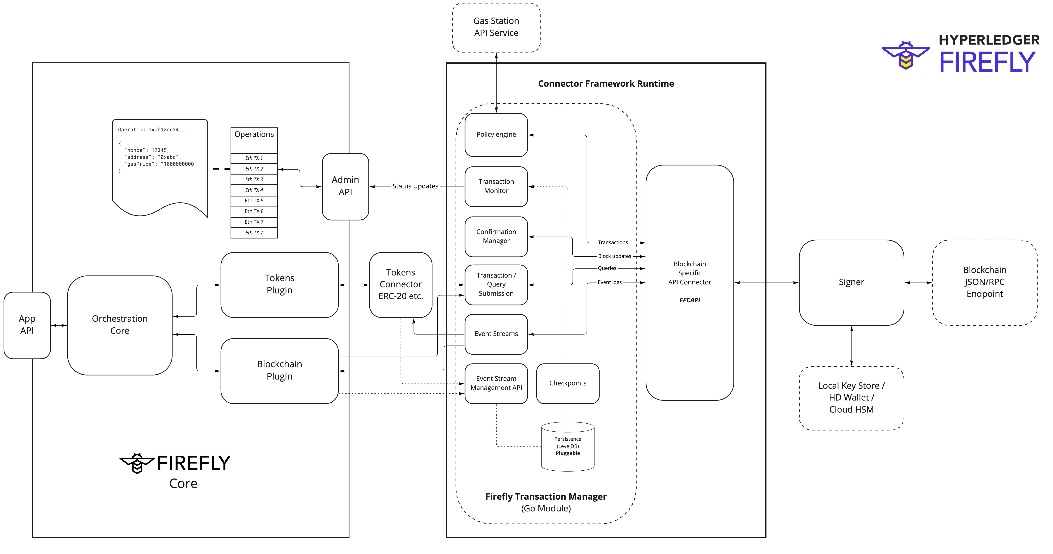
Hyperledger Fabric is intended as a foundation for developing applications or solutions with a modular architecture. Hyperledger Fabric allows components, such as consensus and membership services, to be plug-and-play. Its modular and versatile design satisfies a broad range of industry use cases. It offers a unique approach to consensus that enables performance at scale while preserving privacy.



## Fabconnect:

A reliable REST and WebSocket API to interact with a Fabric network and stream events.

We use fabconnect to stream data from fabric blockchain to firefly UI.



## Firefly SDK:

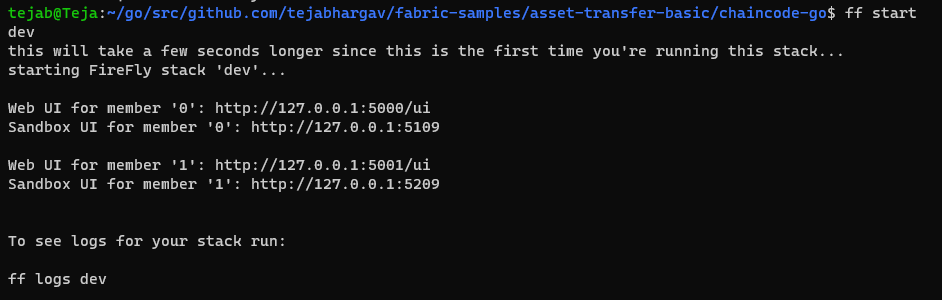
We use firefly SDK in the for of Node modules and import them to our type script frontend to use the build in apis and functions that very helpful for integrating on top of firefly . And it increases the functionality of web sockets which play key role in displaying information through firefly.

Project workflow:

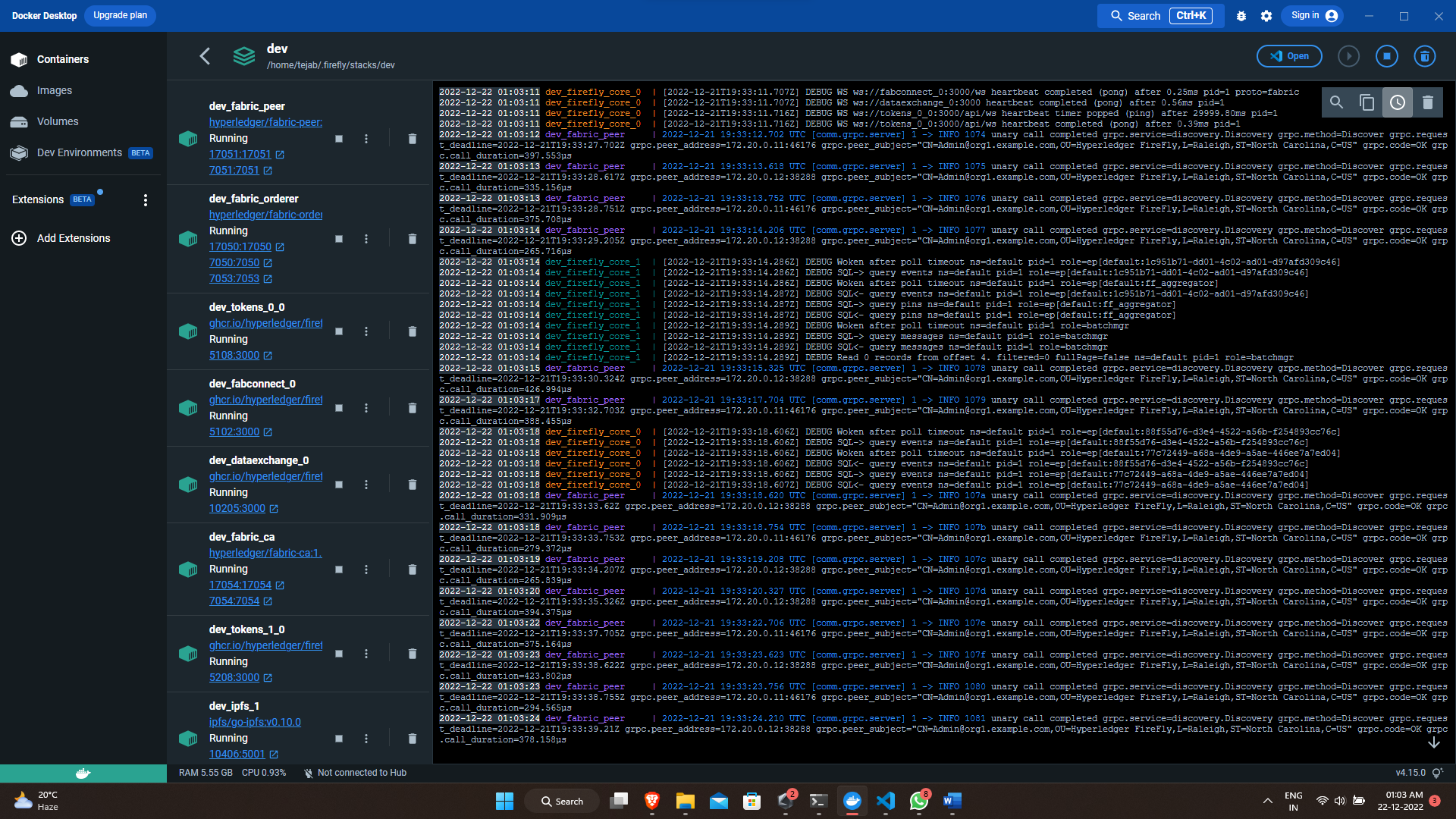
* The front-end UI created using NodeJS and firefly SDK creates events that are streamed to the firefly contract interface.
* Contract Interface invokes the smart contract that is deployed on block chain and interacts with it methods and functions in response to the events generated.
* The Contract interface is created by defining a Json datatype and broadcasting to all the members and organizations in the network so they the super node for the member understands how to interact with the smart contract.
* A Swagger UI is generated in response to the declaration of the Firefly Contract Interface.
* After invoking the smart contracts there are Listeners that are installed on firefly to listen to events that are happening in the blockchain.
* For example if we send a post to create an asset the listeners try to listen to an event called AssetCreated from the blockchain.
* And when there is an event occurring in the blockchain firefly listens and forward them through subscriptions that are created with receive the data in the form of websockets.

Screenshots of entire project:

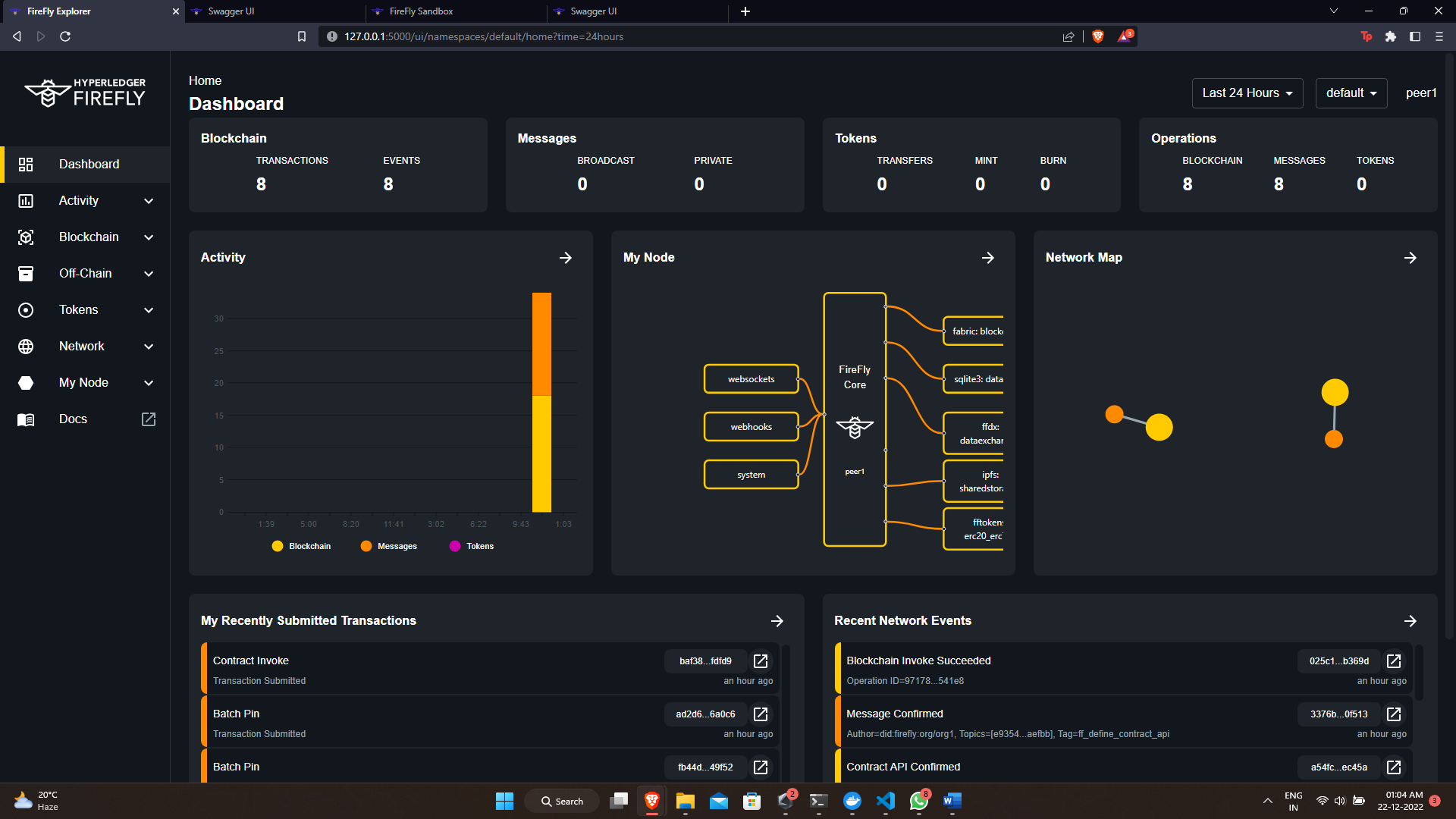
Starting firefly stack :



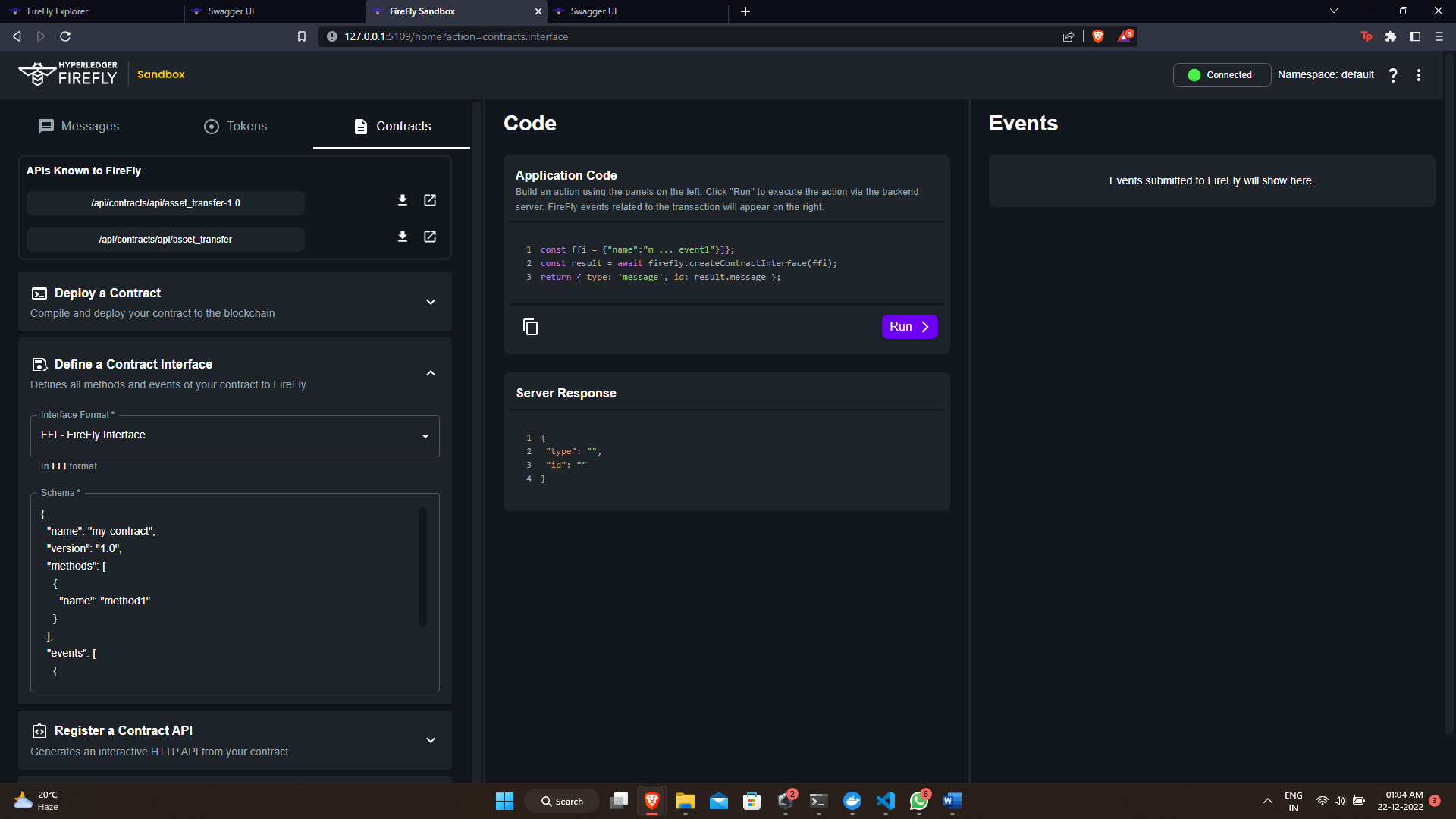
Docker compose that runs both firefly and fabric:



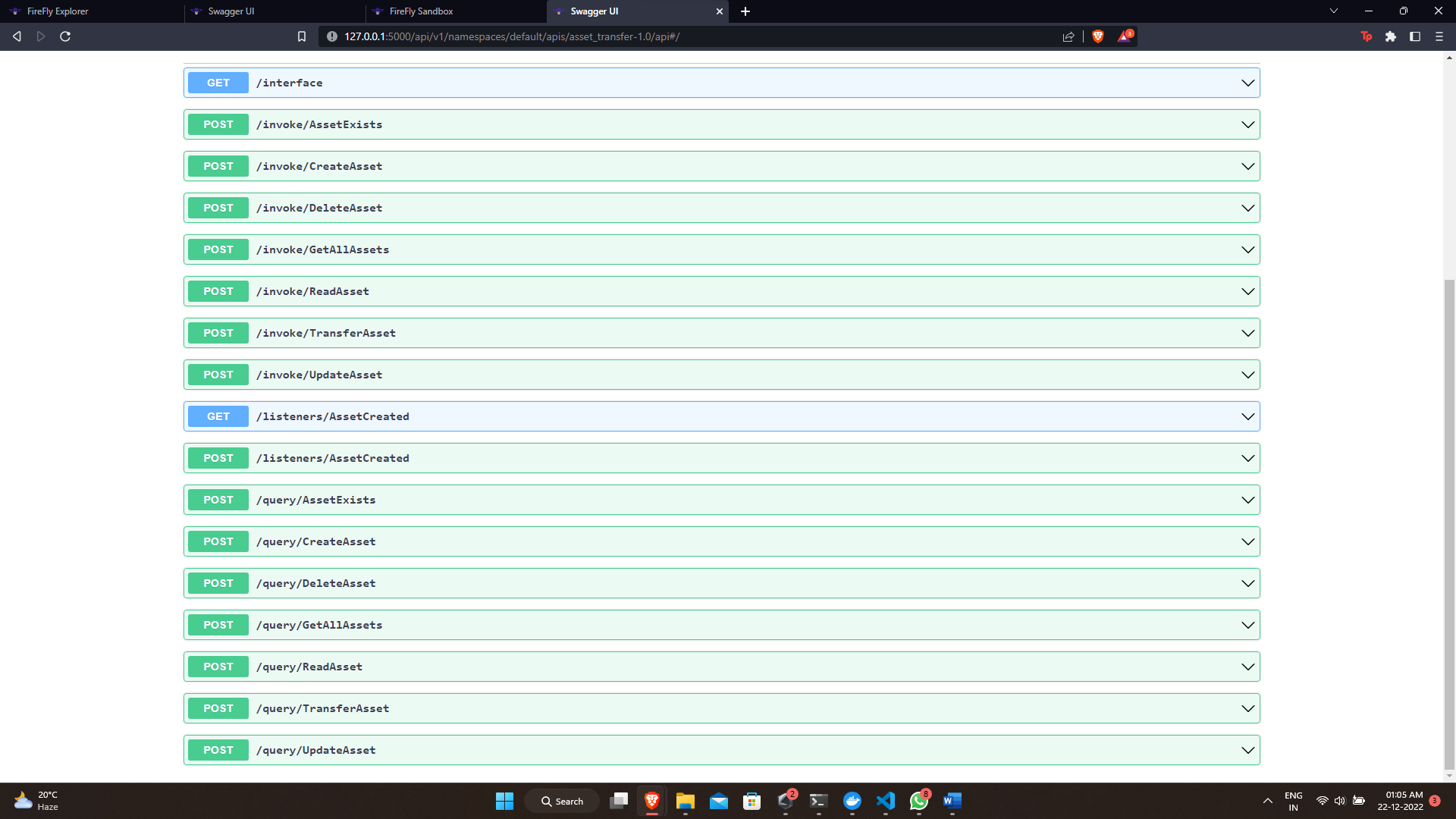
Firefly explorer:



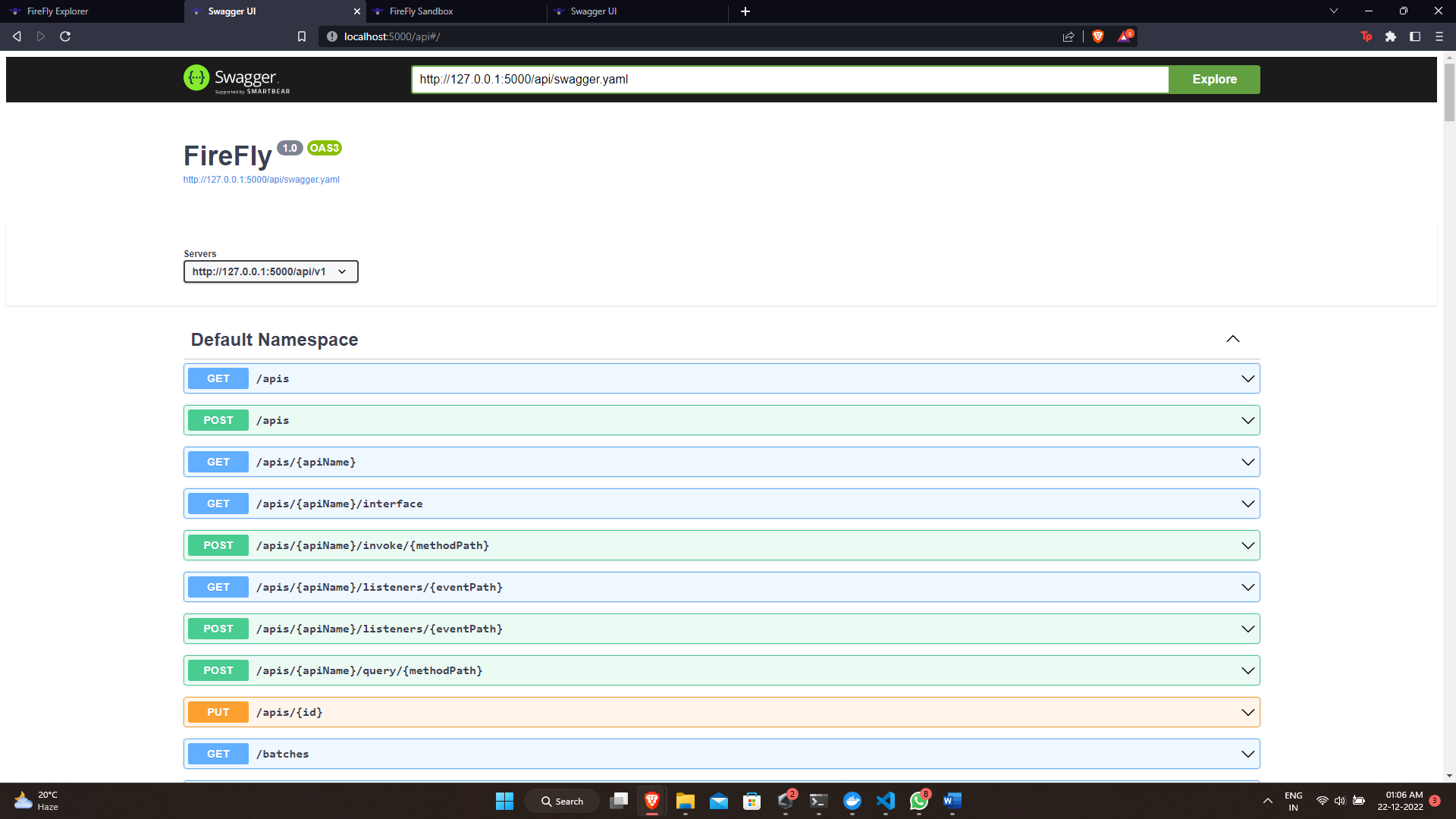
Firefly Sandbox:



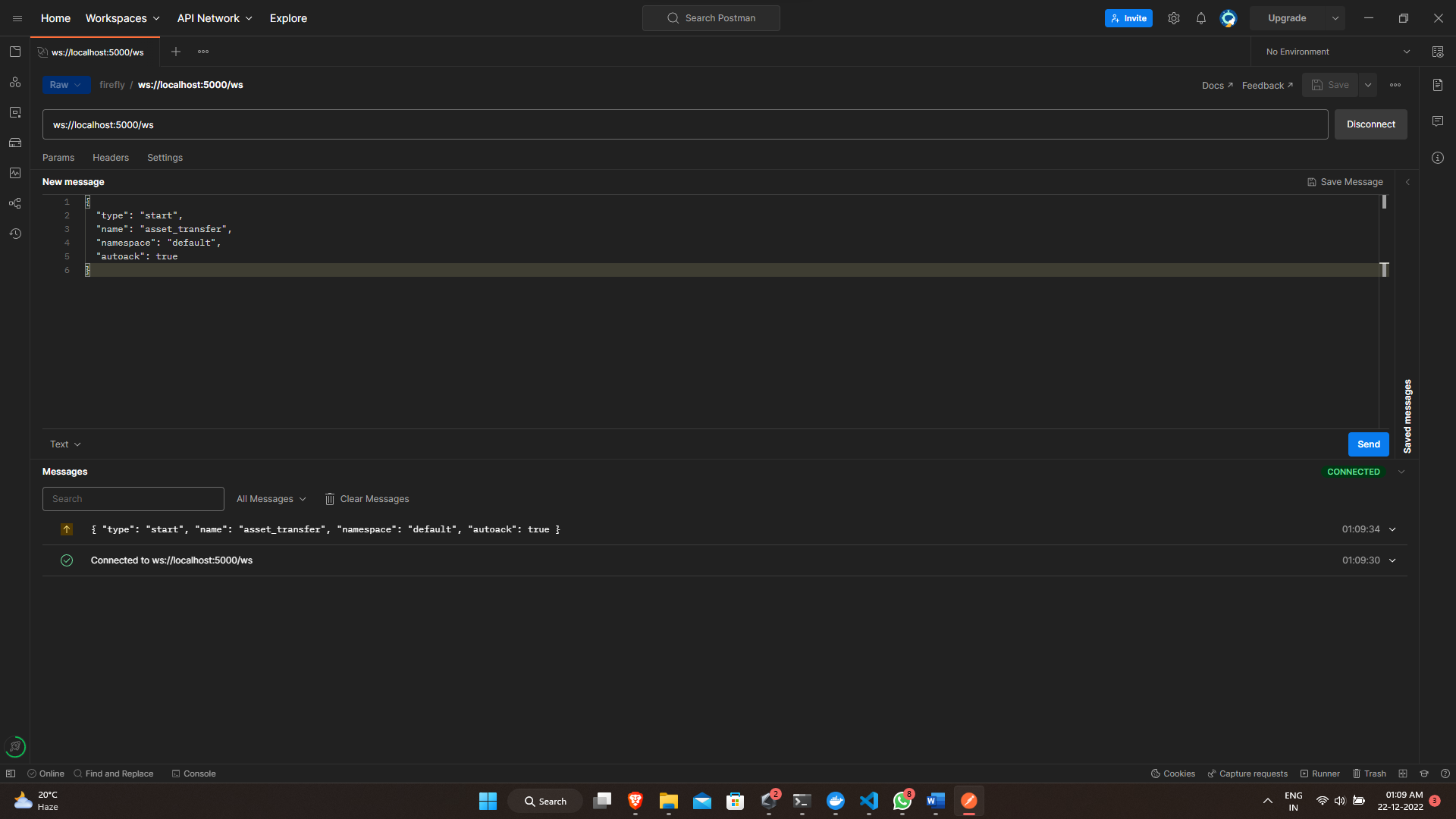
Swagger UI for Contract Interface:



Swagger UI default:



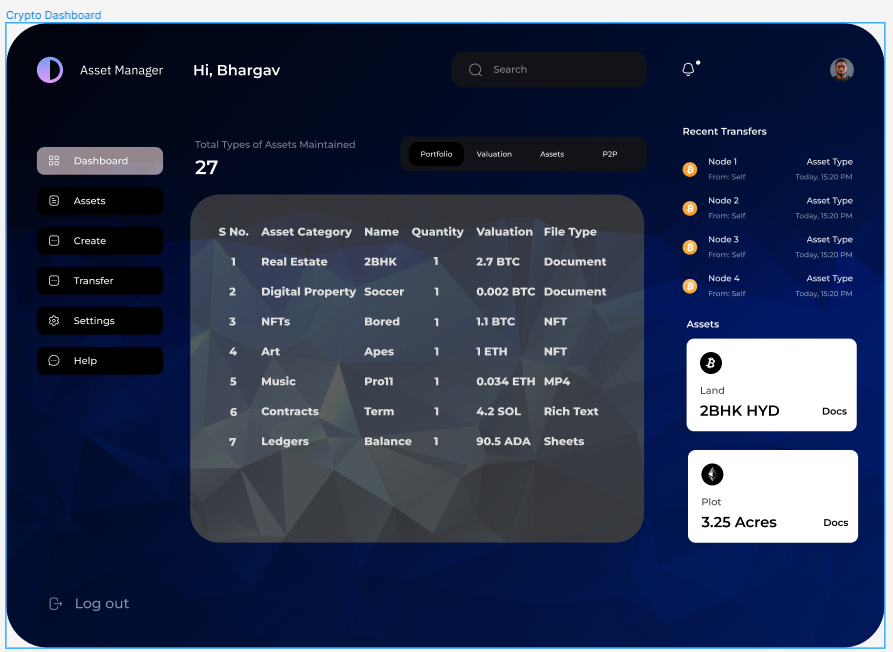
Postman to display webhooks:



Future enhancement plan:

* Create more functionality to the smart contract.
* Development of front-end
* Login authentication and user management
* Integration with the firefly

Front-end UI figma still in development.



## Smartcontract:

package chaincode

import (

    "encoding/json"

    "fmt"

    "github.com/hyperledger/fabric-contract-api-go/contractapi"

)

// SmartContract provides functions for managing an Asset

type SmartContract struct {

    contractapi.Contract

}

// Property describes basic details of what makes up a simple Property

// Insert struct field in alphabetic order => to achieve determinism accross languages

// golang keeps the order when marshal to json but doesn't order automatically

type RealEstateProperty struct {

    Description string `json:"Description"`

    ID          string `json:"ID"`

    Owner       string `json:"Owner"`

    Price       int    `json:"Price"`

    Status      string `json:"Status"`

}

// InitLedger adds a base set of Properties to the ledger

func (s \*SmartContract) InitLedger(ctx contractapi.TransactionContextInterface) error {

    assets := []RealEstateProperty{

        {ID: "asset1", Owner: "Teja", Description: "200 sq yards in Hyderabad", Price: 50000000, Status: "For Sale"},

        {ID: "asset2", Owner: "Bhargav", Description: "200 sq yards in Kolkata", Price: 50000000, Status: "For Sale"},

        {ID: "asset3", Owner: "Smayan", Description: "500 sq yards in Andhra Pradesh", Price: 10000000, Status: "For Sale"},

        {ID: "asset4", Owner: "Prajakta", Description: "200 sq yards in Karnataka", Price: 1000000, Status: "For Sale"},

        {ID: "asset5", Owner: "Resha", Description: "300 sq yards in Maharashtra", Price: 15000000, Status: "For Sale"},

        {ID: "asset6", Owner: "Sahithi", Description: "4bhk in Mumbai", Price: 1500000, Status: "For Sale"},

        {ID: "asset7", Owner: "Rahul", Description: "2bhk in Kerala", Price: 15000000, Status: "For Sale"},

        {ID: "asset8", Owner: "Ujwala", Description: "600 sq ft Flat in AndhraPradhesh", Price: 150000000, Status: "For Sale"},

    }

    for \_, asset := range assets {

        assetJSON, err := json.Marshal(asset)

        if err != nil {

            return err

        }

        err = ctx.GetStub().PutState(asset.ID, assetJSON)

        if err != nil {

            return fmt.Errorf("failed to put to world state. %v", err)

        }

    }

    return nil

}

// CreateProperty issues a new asset to the world state with given details.

func (s \*SmartContract) CreateProperty(ctx contractapi.TransactionContextInterface, id string, owner string, description string, price int, status string) error {

    exists, err := s.PropertyExists(ctx, id)

    if err != nil {

        return err

    }

    if exists {

        return fmt.Errorf("the property %s already exists", id)

    }

    asset := RealEstateProperty{

        ID:          id,

        Owner:       owner,

        Description: description,

        Price:       price,

        Status:      status,

    }

    assetJSON, err := json.Marshal(asset)

    if err != nil {

        return err

    }

    err = ctx.GetStub().PutState(id, assetJSON)

    if err != nil {

        return err

    }

    return ctx.GetStub().SetEvent("PropertyCreated", assetJSON)

}

// ReadProperty returns the asset stored in the world state with given id.

func (s \*SmartContract) ReadProperty(ctx contractapi.TransactionContextInterface, id string) (\*RealEstateProperty, error) {

    assetJSON, err := ctx.GetStub().GetState(id)

    if err != nil {

        return nil, fmt.Errorf("failed to read from world state: %v", err)

    }

    if assetJSON == nil {

        return nil, fmt.Errorf("the asset %s does not exist", id)

    }

    var asset RealEstateProperty

    err = json.Unmarshal(assetJSON, &asset)

    if err != nil {

        return nil, err

    }

    return &asset, nil

}

// UpdateProperty updates an existing asset in the world state with provided parameters.

func (s \*SmartContract) UpdateProperty(ctx contractapi.TransactionContextInterface, id string, owner string, description string, price int, status string) error {

    exists, err := s.PropertyExists(ctx, id)

    if err != nil {

        return err

    }

    if !exists {

        return fmt.Errorf("the property %s does not exist", id)

    }

    // overwriting original property with new property

    asset := RealEstateProperty{

        ID:          id,

        Owner:       owner,

        Description: description,

        Price:       price,

        Status:      status,

    }

    assetJSON, err := json.Marshal(asset)

    if err != nil {

        return err

    }

    return ctx.GetStub().PutState(id, assetJSON)

}

// DeleteProperty deletes an given property from the world state.

func (s \*SmartContract) DeleteProperty(ctx contractapi.TransactionContextInterface, id string) error {

    exists, err := s.PropertyExists(ctx, id)

    if err != nil {

        return err

    }

    if !exists {

        return fmt.Errorf("the property %s does not exist", id)

    }

    return ctx.GetStub().DelState(id)

}

// PropertyExists returns true when property with given ID exists in world state

func (s \*SmartContract) PropertyExists(ctx contractapi.TransactionContextInterface, id string) (bool, error) {

    assetJSON, err := ctx.GetStub().GetState(id)

    if err != nil {

        return false, fmt.Errorf("failed to read from world state: %v", err)

    }

    return assetJSON != nil, nil

}

// TransferProperty updates the owner field of property with given id in world state.

func (s \*SmartContract) TransferProperty(ctx contractapi.TransactionContextInterface, id string, newOwner string) (string, error) {

    asset, err := s.ReadProperty(ctx, id)

    if err != nil {

        return "", err

    }

    oldOwner := asset.Owner

    asset.Owner = newOwner

    assetJSON, err := json.Marshal(asset)

    if err != nil {

        return "", err

    }

    err = ctx.GetStub().PutState(id, assetJSON)

    if err != nil {

        return "", err

    }

    return oldOwner, nil

}

// GetAllProperties returns all properties found in world state

func (s \*SmartContract) GetAllProperties(ctx contractapi.TransactionContextInterface) ([]\*RealEstateProperty, error) {

    // range query with empty string for startKey and endKey does an

    // open-ended query of all assets in the chaincode namespace.

    resultsIterator, err := ctx.GetStub().GetStateByRange("", "")

    if err != nil {

        return nil, err

    }

    defer resultsIterator.Close()

    var assets []\*RealEstateProperty

    for resultsIterator.HasNext() {

        queryResponse, err := resultsIterator.Next()

        if err != nil {

            return nil, err

        }

        var asset RealEstateProperty

        err = json.Unmarshal(queryResponse.Value, &asset)

        if err != nil {

            return nil, err

        }

        assets = append(assets, &asset)

    }

    return assets, nil

}

## Contract Interface:

{

  "namespace": "default",

  "name": "asset\_transfer",

  "description": "Spec interface for the asset-transfer-basic golang chaincode",

  "version": "1.0",

  "methods": [

    {

      "name": "GetAllAssets",

      "pathname": "",

      "description": "",

      "params": [],

      "returns": [

        {

          "name": "",

          "schema": {

            "type": "array",

            "details": {

              "type": "object",

              "properties": {

                "type": "string"

              }

            }

          }

        }

      ]

    },

    {

      "name": "CreateAsset",

      "pathname": "",

      "description": "",

      "params": [

        {

          "name": "id",

          "schema": {

            "type": "string"

          }

        },

        {

          "name": "owner",

          "schema": {

            "type": "string"

          }

        },

        {

          "name": "description",

          "schema": {

            "type": "string"

          }

        },

        {

          "name": "price",

          "schema": {

            "type": "string"

          }

        },

        {

          "name": "status",

          "schema": {

            "type": "string"

          }

        }

      ],

      "returns": []

    },

    {

      "name": "ReadAsset",

      "pathname": "",

      "description": "",

      "params": [

        {

          "name": "id",

          "schema": {

            "type": "string"

          }

        }

      ],

      "returns": []

    },

    {

      "name": "UpdateAsset",

      "pathname": "",

      "description": "",

      "params": [

        {

          "name": "id",

          "schema": {

            "type": "string"

          }

        },

        {

          "name": "owner",

          "schema": {

            "type": "string"

          }

        },

        {

          "name": "description",

          "schema": {

            "type": "string"

          }

        },

        {

          "name": "price",

          "schema": {

            "type": "string"

          }

        },

        {

          "name": "status",

          "schema": {

            "type": "string"

          }

        }

      ],

      "returns": []

    },

    {

      "name": "DeleteAsset",

      "pathname": "",

      "description": "",

      "params": [

        {

          "name": "id",

          "schema": {

            "type": "string"

          }

        }

      ],

      "returns": []

    },

    {

      "name": "AssetExists",

      "pathname": "",

      "description": "",

      "params": [

        {

          "name": "id",

          "schema": {

            "type": "string"

          }

        }

      ],

      "returns": []

    },

    {

      "name": "TransferAsset",

      "pathname": "",

      "description": "",

      "params": [

        {

          "name": "id",

          "schema": {

            "type": "string"

          }

        },

        {

          "name": "newOwner",

          "schema": {

            "type": "string"

          }

        }

      ],

      "returns": []

    }

  ],

  "events": [

    {

      "name": "AssetCreated"

    }

  ]

}