**Test document**

We generated 8 different chaincodes by using ChatGPT. Below are the chaincodes-

1. DEX AMM
2. DEX Orderbook
3. First Price Bid
4. Mint NFT
5. NFT Collectibles
6. NFT Event Ticket
7. NFT Real Estate
8. NFT Second Price Bid

The modifications applied to the generated code included the following actions:

* Altering the package name.
* Disabling the main functions by commenting them out.
* Adjusting the SmartContract function.

After fixing these codes, they were run with the “[fabric samples](https://github.com/hyperledger/fabric-samples)” code and were run perfectly. For this, we followed the following steps taken from [this link](https://hyperledger-fabric.readthedocs.io/en/release-2.5/).

In order to successfully run the "[fabric samples](https://github.com/hyperledger/fabric-samples)" code, we implemented a series of corrective measures outlined in the steps provided by the [Hyperledger Fabric Docs](https://hyperledger-fabric.readthedocs.io/en/release-2.5/). These adjustments ensured the smooth execution of the program.

* To begin, create a folder named "Projects" within your home directory and then navigate into this newly created folder.
* mkdir Projects
* cd Projects
* Next, execute the specified command from the terminal while you are inside the "Projects" folder.
* curl -sSL https://bit.ly/2ysbOFE | bash -s
* Once the script completes its execution, a new folder named "fabric-samples" will be created. You should then navigate to the "test-network" folder within this "fabric-samples" directory.
* cd test-network
* To clean up any containers or artifacts from previous runs, execute the provided command in the terminal while in the "test-network" folder. This will ensure a fresh environment for your current operation.
* ./network.sh down
* To initiate the network, issue the given command while in the "test-network" folder. This command will start the network setup process.
* ./network.sh up
* To create a channel with the default name "mychannel," execute the specified command while you are in the "test-network" folder. This command will initiate the creation of the channel.
* ./network.sh createChannel
* To create a channel with a custom name, use the channel flag (-c) followed by your desired channel name when running the command. This allows for the creation of a channel with a name different from the default "mychannel."
* ./network.sh createChannel -c channel1
* To simultaneously bring up the network and create a channel in one step, use a combined command that initiates the network setup and specifies the channel creation, either with the default name "mychannel" or with a custom name using the channel flag (-c). This streamlines the process, allowing both actions to be completed together.
* ./network.sh up createChannel
* Once you have completed your tasks with the test network, you can bring down the network by executing the provided command. This will terminate the network setup and clean up any resources it was using.
* ./network.sh down
* To start a chaincode on the channel, execute the specified command. This action will deploy the chaincode onto the channel you've previously created or specified in the test network.
* ./network.sh deployCC -ccn basic -ccp ../asset-transfer-basic/chaincode-go -ccl go
* In this context, "../asset-transfer-basic/chaincode-go" refers to the relative path of the chaincode, as seen from the folder containing the "network.sh" file, with 'ccp' signifying the chaincode place. To run a different chaincode, simply alter the directory in the command to point to the new chaincode's location. This flexibility allows for easy switching between different chaincodes as needed.

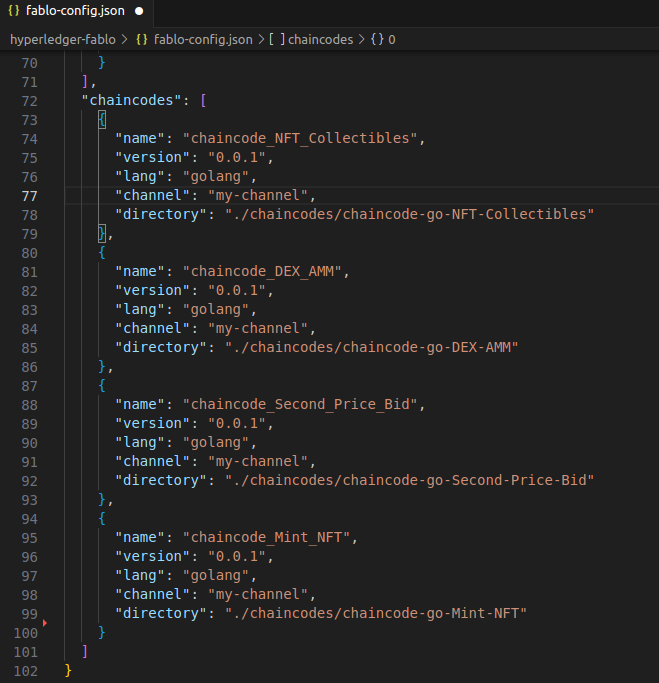
The successful deployment and execution of all eight chaincodes using the described process marks a significant milestone. However, to verify their operational effectiveness, it's essential to test them by invoking various methods within each chaincode. For this purpose, the tool "[Fablo](https://github.com/hyperledger-labs/fablo)" was employed. The subsequent steps were taken to facilitate the testing of the chaincodes using Fablo.

* To begin, create a folder named "hyperledger-fablo" within the existing "fabric-samples" directory. After creating this folder, navigate inside it to proceed with the next steps.
* mkdir hyperledger-fablo
* cd hyperledger-fablo
* Next, execute the specified command from the terminal while you are within the "hyperledger-fablo" folder. This step is part of the process to set up or configure the environment using Fablo.
* curl -Lf https://github.com/hyperledger-labs/fablo/releases/download/1.2.0/fablo.sh -o ./fablo && chmod +x ./fablo
* After running the previous command, a single shell script named "fablo" will be generated. To create a local Hyperledger Fabric network with Node.js chaincode and a REST API client, you need to install Fablo. Once Fablo is installed, execute the given command. This process will set up your desired network configuration using Fablo.
* ./fablo init node rest
* ./fablo up
* Once you have executed the command to set up the network using Fablo, it will take a few minutes for the entire network to be fully established and operational. When you are finished using the network or wish to shut it down, you can do so by running the specified command. This will bring down the network and terminate all its associated processes.
* ./fablo down
* To completely destroy the network, execute the provided command. This action will dismantle the network setup and remove all related configurations and data, effectively resetting the environment to its state before the network was established.
* ./fablo prune
* Following the execution of the initial command (`./fablo init node rest`), two key items are generated:

A file named `fablo-config.json`: This configuration file outlines the network topology, including details about the root organization, other participating organizations, channels, and chaincodes.

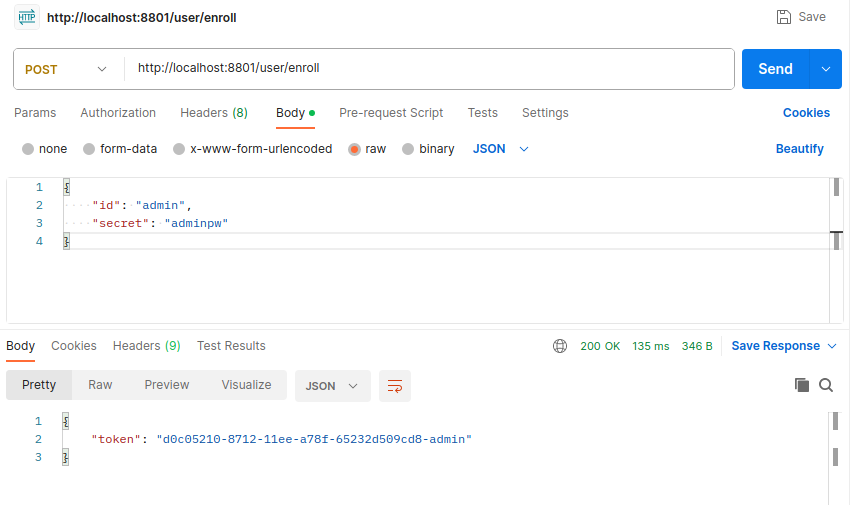
A folder named `chaincodes`: This is the designated location for copying and pasting your generated chaincodes.

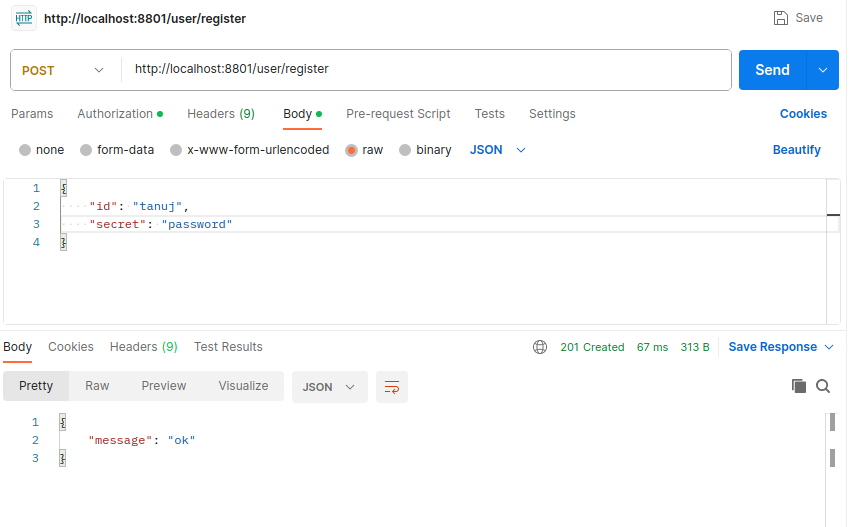
* It's important to note that the generated `fablo-config.json` file is configured to use a single node Solo consensus mechanism and does not include support for TLS (Transport Layer Security). This setup defines the basic structure and security features of the Hyperledger Fabric network you are working with.
* To customize your Hyperledger Fabric network, you can modify the `fablo-config.json` file. Specifically, within this JSON file, there is a field named `chaincodes`, which is structured as an array. This array allows you to specify one or more chaincodes to be installed when the fabric network is initialized. The configuration of this array will dictate which chaincodes are available on your network. For a clear understanding, a screenshot or example of how this array is structured would be helpful.



* Once you have made the necessary modifications to the `fablo-config.json` file, you can recreate the network to reflect these changes. To do this, run the specified command. This will initiate the process of setting up the network again, this time using the updated configuration in the `fablo-config.json` file.
* ./fablo recreate fablo-config.json
* Following the recreation of the network with the updated `fablo-config.json` file, you are now ready to test your chaincodes. To do this, you'll use specific commands designed to interact with and test each chaincode. These commands will help you verify the functionality and performance of the chaincodes within your Hyperledger Fabric network environment.
* chaincode\_NFT\_Collectibles - docker exec cli.org1.example.com peer chaincode invoke peer0.org1.example.com -C "my-channel" -n "chaincode\_NFT\_Collectibles" -c '{"Args":["SmartContract:InitLedger"]}'
* chaincode\_DEX\_AMM - docker exec cli.org1.example.com peer chaincode invoke peer0.org1.example.com -C "my-channel" -n "chaincode\_DEX\_AMM" -c '{"Args":["SmartContract:InitLedger"]}'
* chaincode\_Second\_Price\_Bid - docker exec cli.org1.example.com peer chaincode invoke peer0.org1.example.com -C "my-channel" -n "chaincode\_Second\_Price\_Bid" -c '{"Args":["SmartContract:RevealWinner", "1234"]}'
* chaincode\_Mint\_NFT - docker exec cli.org1.example.com peer chaincode invoke peer0.org1.example.com -C "my-channel" -n "chaincode\_Mint\_NFT" -c '{"Args":["SmartContract:InitLedger"]}'
* The testing process confirmed that four of the chaincodes functioned correctly. However, when attempting to invoke methods in the remaining four chaincodes, we encountered errors.
* Error: endorsement failure during invoke. response: status:500 message:"error in simulation: failed to execute transaction<transaction\_number>: could not launch chaincode <chaincode\_name>.0.1:<transaction\_number>: chaincode registration failed: container exited with 2"
* Upon thorough examination of the four chaincodes that encountered errors, we identified several key similarities unique to them, which were not observed in the successfully tested chaincodes. These similarities include the following points:
  + *stub shim.ChaincodeStubInterface* instead of *ctx contractapi.TransactionContextInterface*
  + *return shim.Success(nil)* instead of *return nil*
  + *pb.Response* instead of *error*
  + *shim.Error* instead of *fmt.Errorf*
* The analysis revealed that all the chaincodes generating errors were utilizing two common libraries from GitHub, named `shim` and `peer`. These libraries are no longer used in the latest versions of Hyperledger Fabric. Since our Fablo network operates on the most recent Hyperledger Fabric code, it is unable to invoke the chaincodes that depend on these outdated and incompatible libraries.
* The positive aspect of this situation is that we have the option to update the outdated chaincodes. This involves modifying them to replace the older `shim` and `peer` libraries with their newer counterparts that are compatible with the latest versions of Hyperledger Fabric. Such an update will ensure that these chaincodes can function properly in the current Fablo network environment.
* The "chaincode\_NFT\_Real\_Estate" was updated to integrate modern libraries, effectively replacing the older `shim` and `peer` versions, and this adjustment led to its successful operation in the latest Hyperledger Fabric environment on our Fablo network.
* All the codes will be found in this [GitHub repo](https://github.com/tanujsurve/Hyperledger_chaincode). The fixed chaincode\_NFT\_Real\_Estate chaincode can be found in [this commit](https://github.com/tanujsurve/Hyperledger_chaincode/commit/fae0c1891bd8da75ff755198d08fee96b829b72d). The previous code can be found in the [previous commit](https://github.com/tanujsurve/Hyperledger_chaincode/commit/20a392dcf7b82d43be3f7c76a155acc2e0ed15ff#diff-1f6cc1df4bdd81a469ba77945c615cd3d8140b9858e182957393a6c1cba5fa13)

**Authorization**

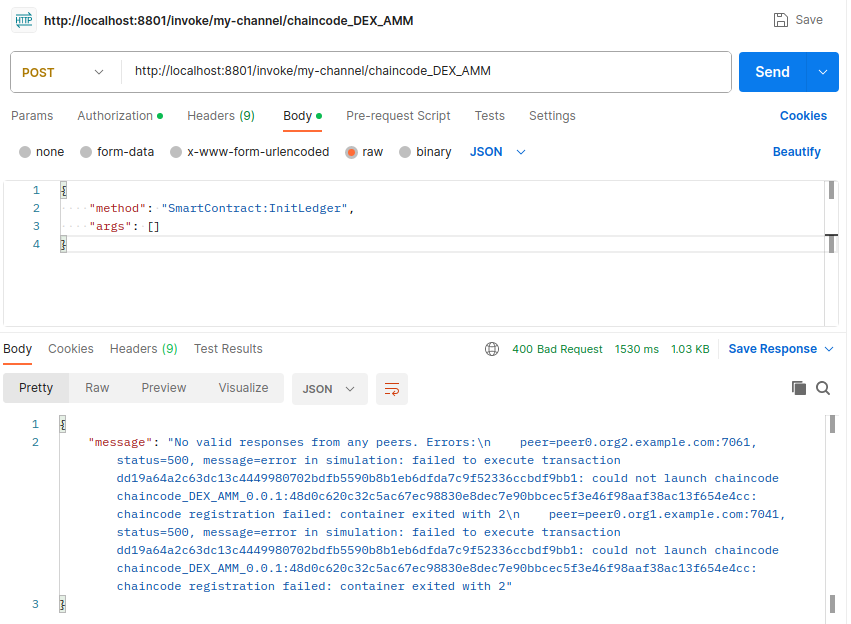
* After the network is recreated, we can test our chaincodes now. To test the chaincodes one by one, we will use the REST APIs provided by Fablo. We will use [Postman](https://www.postman.com/). By default, the APIs will be available at <http://localhost:8801>.
* To call the APIs, first, we need authentication. We will use the admin user created by default to generate a token. Then we will register a new account by using the admin token. The steps are shown below.



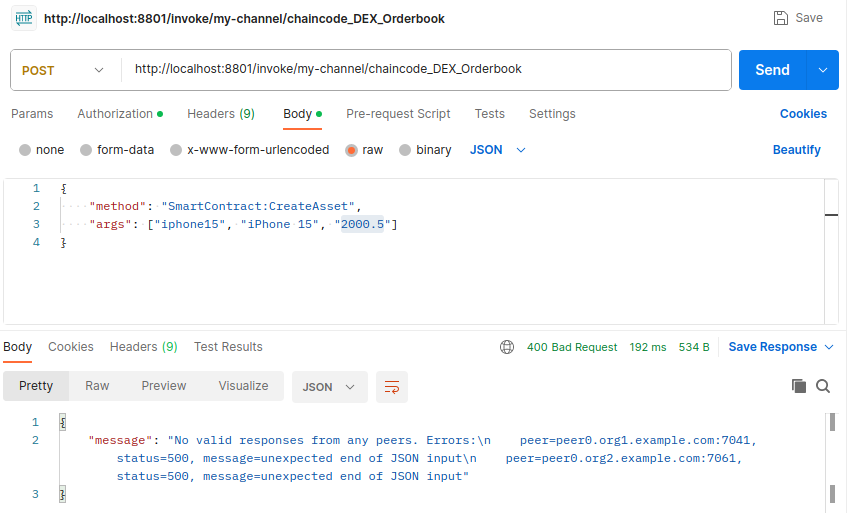
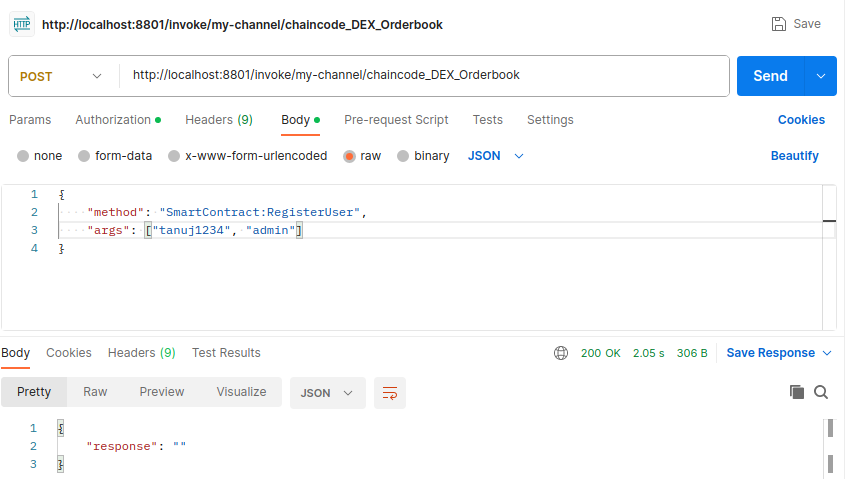
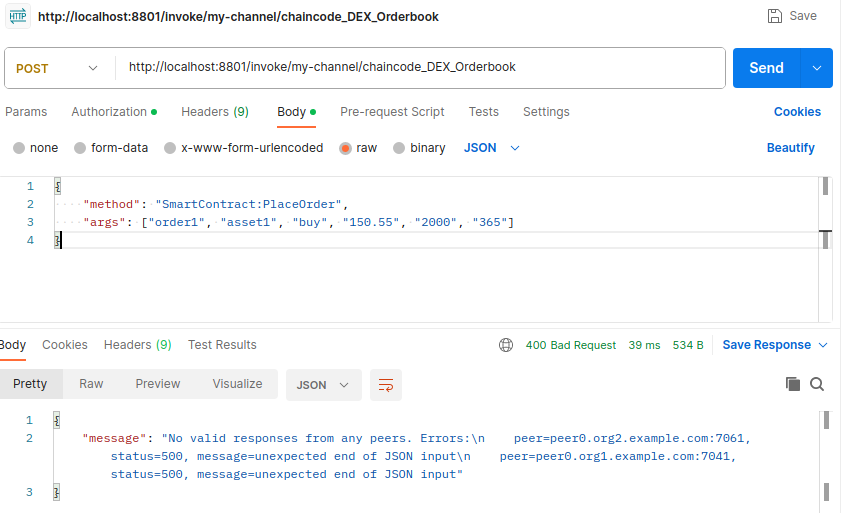
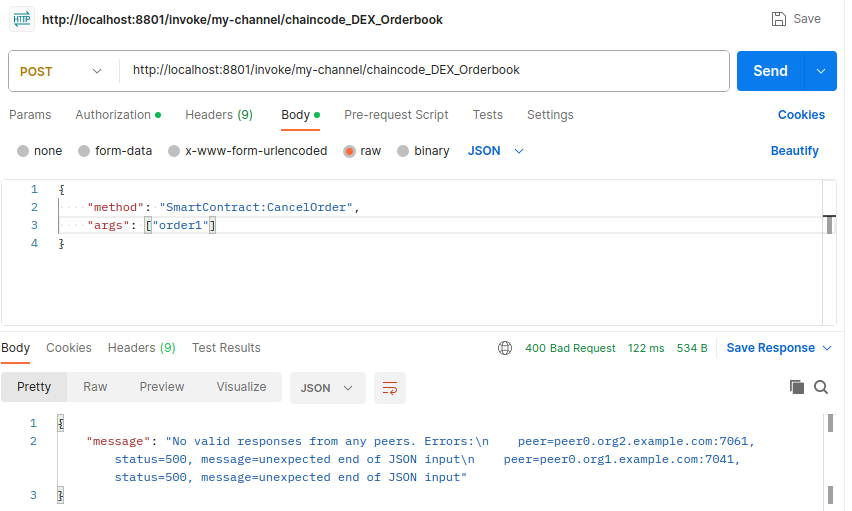
* We will then generate a new token using the newly registered account and use this token to invoke all the methods of the chaincodes.

**DEX AMM**

* Now we will test the first chaincode named **DEX AMM**. It has 4 methods InitLedger, Swap, AddLiquidity, and RemoveLiquidity. First, we will try to invoke the first method InitLedger.
* We can see that invoking the method is producing an error. In fact, invoking all 4 methods of this chaincode gives us the same error saying “*chaincode registration failed: container exited with 2*”.

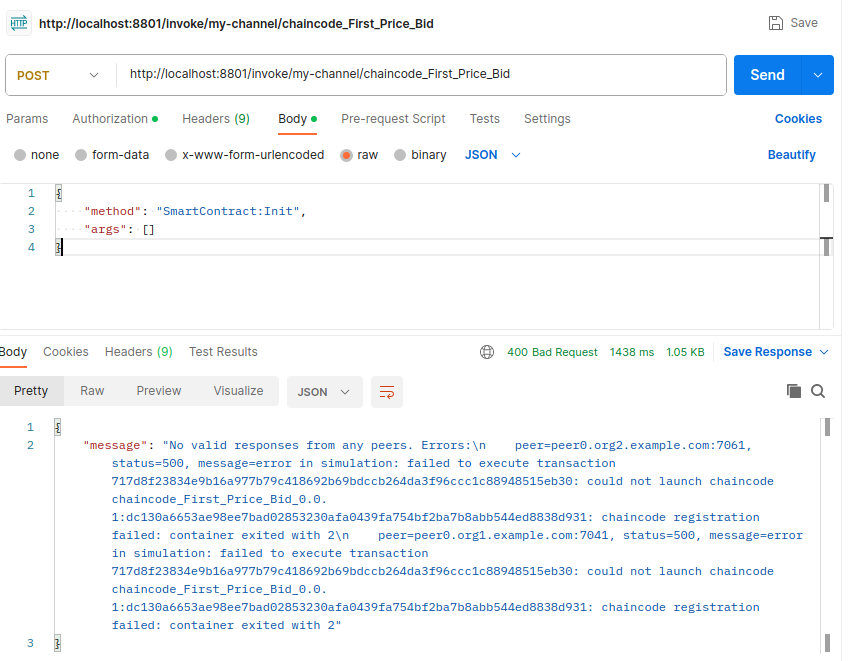


**DEX Orderbook**

* Now let’s test the second chaincode named **DEX Orderbook**. It has 4 methods RegisterUser, CreateAsset, PlaceOrder, and CancelOrder. First, we will try to invoke the first method RegisterUser.
* So, the method worked perfectly. Now let’s create an asset by invoking the CreateAsset method.
* We can see that invoking the method is producing an error saying “*unexpected end of JSON input*”. So, there are bugs in this method.
* As CreateAsset method is not working properly, we can’t create an asset. As a consequence, no *assetID* is available to us which is one of the arguments needed to be passed on to the PlaceOrder method. Thus, invoking the PlaceOrder method without a proper *assetID* will give us errors and it is giving us the same error as we can see in the following picture.
* As PlaceOrder method is not working properly, we can’t place an order for an asset. As a consequence, no *orderID* is available to us which is one of the arguments needed to be passed on to the CancelOrder method. Thus, invoking the CancelOrder method without a proper *orderID* will give us errors and it is giving us the same error as we can see in the previous picture.

**First Price Bid**

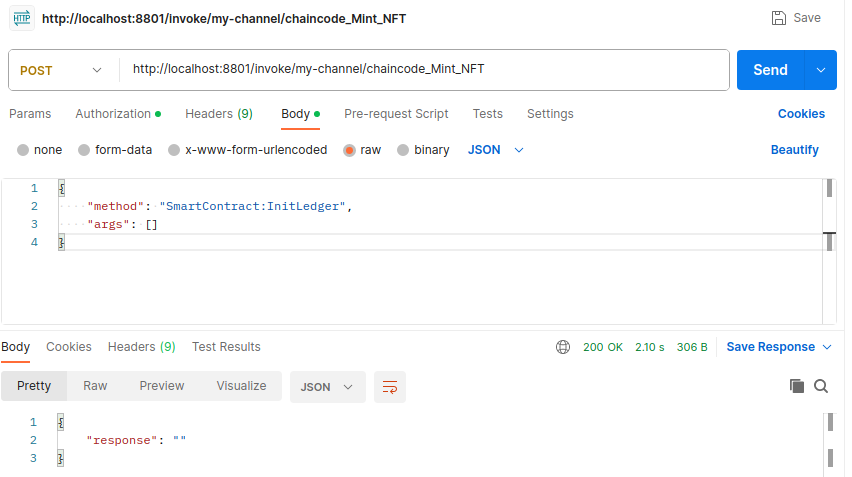
* Now we will test the chaincode named **First Price Bid**. It has 2 methods Init, and Invoke. First, we will try to invoke the first method Init.

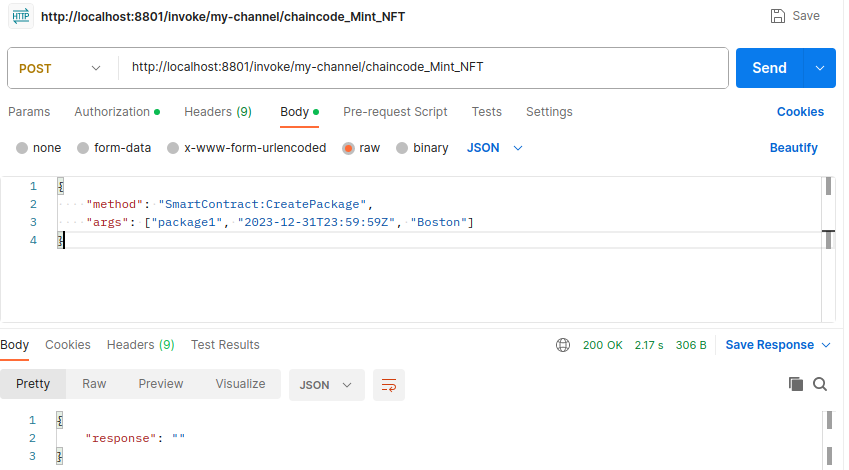


* We can see that invoking the method is producing the same error that we previously encountered with the **DEX AMM** chaincode. Invoking the Invoke method also gives the same error saying “*chaincode registration failed: container exited with 2*”.

**Mint NFT**

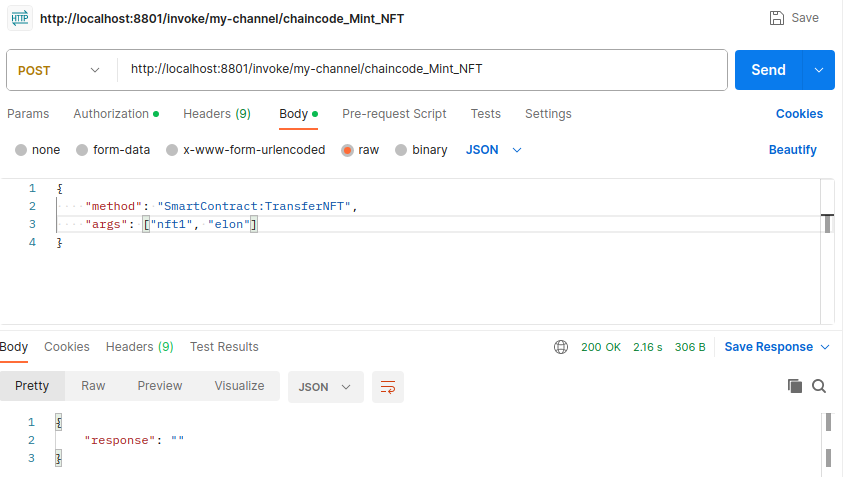
* Now we will test the chaincode named **Mint NFT**. It has 5 methods InitLedger, CreatePackage, MintNFT, TransferNFT, and GetNFT. First, we will try to invoke the first method InitLedger.



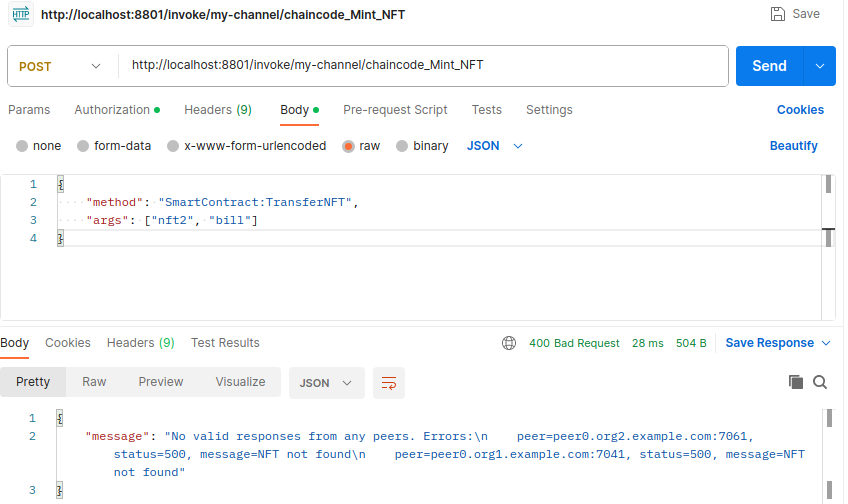
* So, the method worked successfully. Now let’s create a package by invoking the CreatePackage method.
* We have successfully created the “*package1*” package. Now let’s mint an NFT by invoking the MintNFT method.

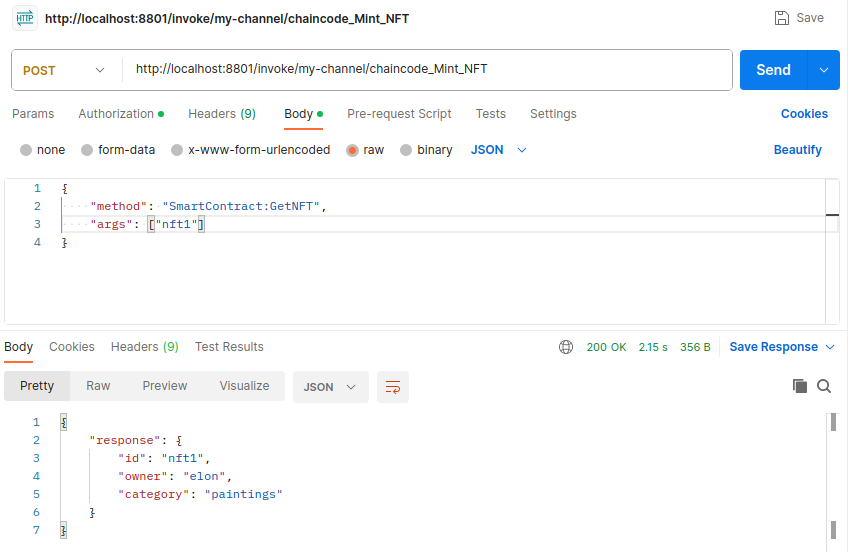


* The MintNFT method created the “*nft1*” NFT. Now let’s transfer this NFT to an owner using the TransferNFT method.



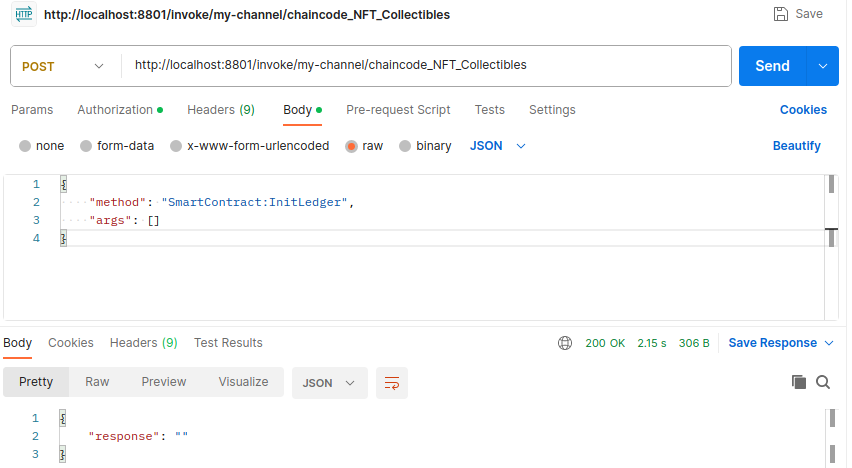
* We can see from the above picture that the TransferNFT method has successfully transferred the “*nft1*” NFT from the previous owner “*tanuj*” to a new owner “*elon*”. What if we try to transfer an NFT that has not been minted yet? Let’s see -

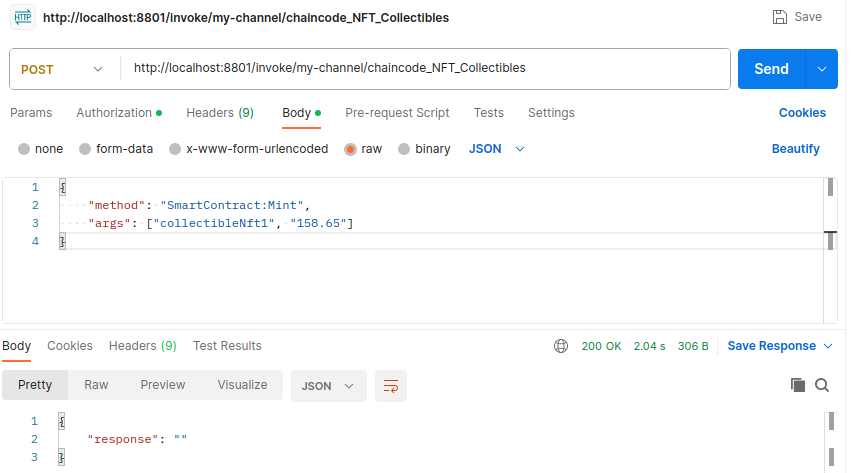


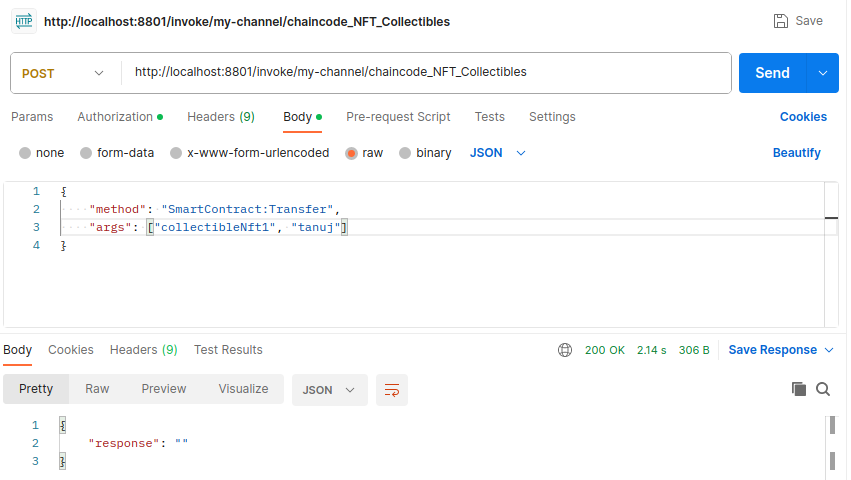
* So, we can see that the chaincode has given an error saying “*NFT not found*”. So, the chaincode logic is working fine.
* Finally, we will invoke the GetNFT method to get details of a minted NFT.
* We can see that the GetNFT method returns the details of the NFT “*nft1*”. We can also see that the owner has been updated according to the change by invoking the TransferNFT method previously.

**NFT Collectibles**

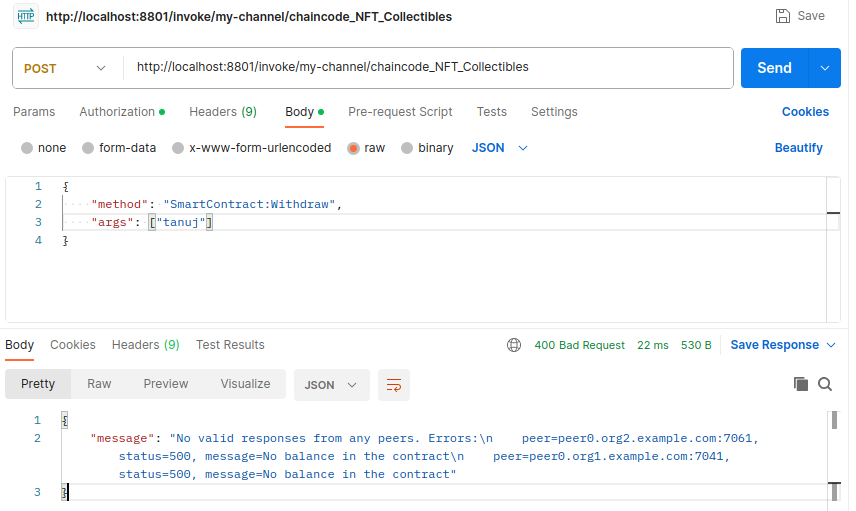
* Now let’s test the **NFT Collectibles** chaincode. It has 4 methods InitLedger, Mint, Transfer, and Withdraw. First, we will try to invoke the first method InitLedger.



* The method worked successfully. Now let’s mint an NFT by invoking the Mint method.
* The “*collectibleNft1*” has been properly minted. Now let’s transfer this NFT to an owner.



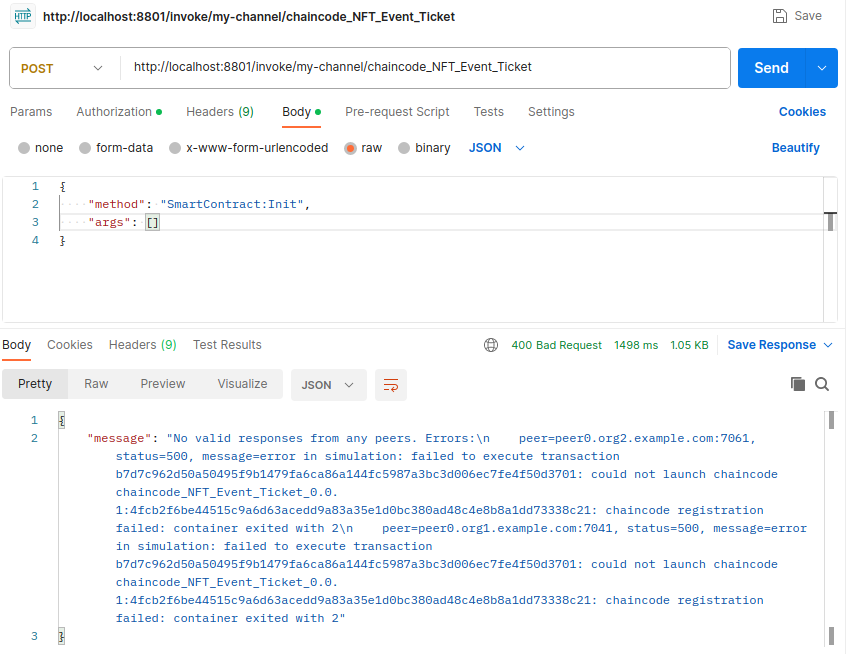
* We can see from the above picture that the Transfer method has successfully transferred the “*collectibleNft1*” NFT to a new owner “*tanuj*”. Now, let’s withdraw the NFT from the owner “*tanuj*” by invoking the Withdraw method.



* Invoking this method gives us an error saying “*No balance in the contract*”. It is because our contract has no balance and there is no method currently present in the chaincode to add balance to the contract.

**NFT Event Ticket**

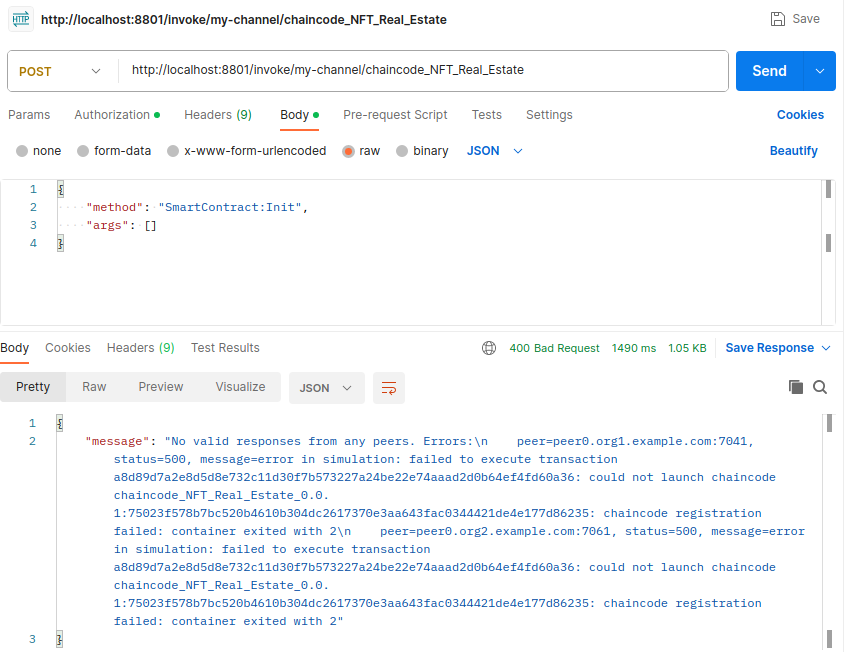
* Next, we will test the chaincode named **NFT Event Ticket**. It has 8 methods Init, Invoke, MintTicket, BuyTicket, RedeemRewardPoints, MarkAttendance, TradeInTicketsForSpecial, and RegisterEvent. First, we will try to invoke the method Init.



* Again, the same error! Invoking all the 7 methods also gives the same error saying “*chaincode registration failed: container exited with 2*”, just like we previously encountered with the **DEX AMM** and **First Price Bid** chaincode.

**NFT Real Estate**

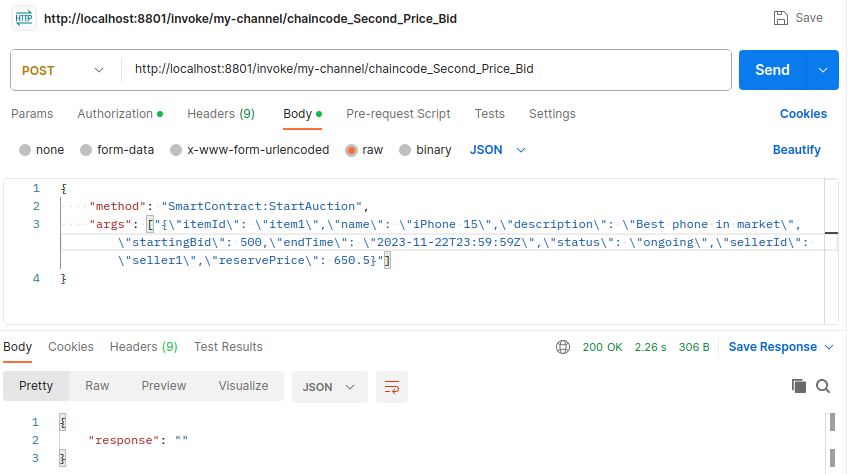
* Now, let’s test the chaincode named **NFT Real Estate**. It has 2 methods Init, and Invoke. First, we will try to invoke the method Init.

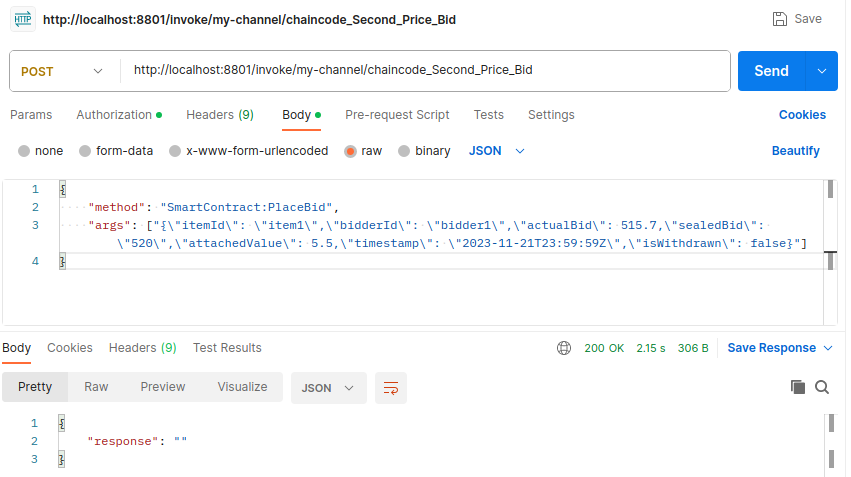


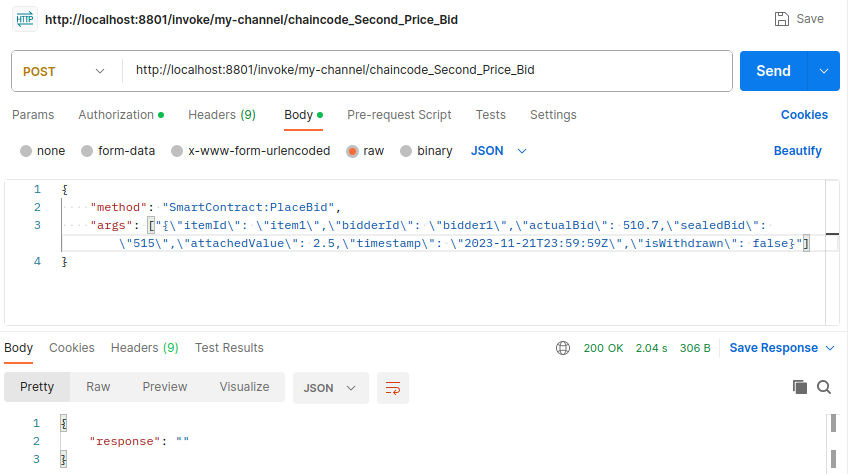
* Alas! We have run into the same error for the fourth time. And, obviously, the Invoke method will also produce the same error.

**NFT Second Price Bid**

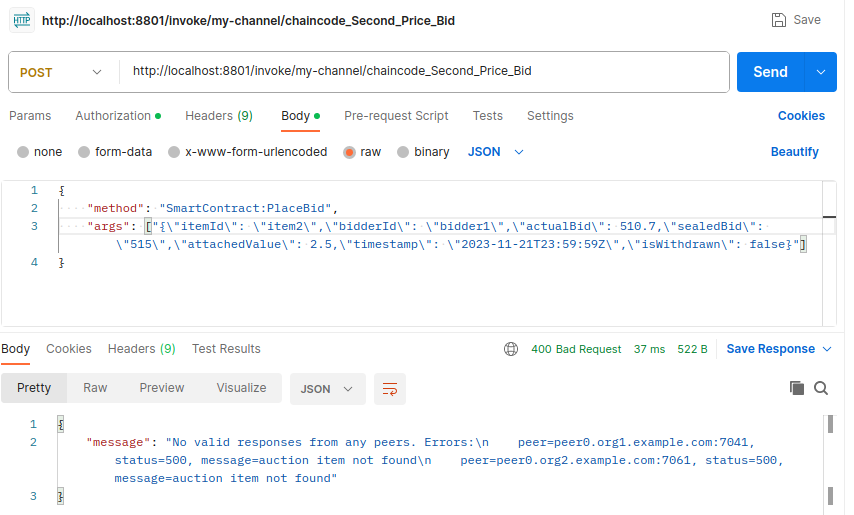
* Now, let’s test the chaincode named **NFT Second Price Bid**. It has 5 methods StartAuction, PlaceBid, EndAuction, RevealWinner, and ClaimNFTAndExcessValue. First, we will try to invoke the method StartAuction.



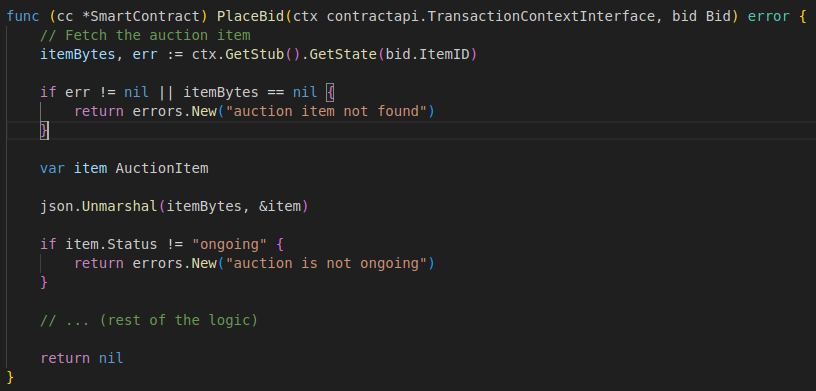
* We have successfully started the auction of the item named “*iPhone 15*” having itemId “*item1*”. One thing to notice is that we passed the “*item1*” object to the chaincode in JSON stringified format. That’s because we can pass only string values into all Hyperledger Fabric chaincodes. Now let’s place some bids for this item by invoking the PlaceBid method.
* So, the “*bidder1*” bidder has successfully bid for the “*item1*” item. Now, let’s add a new bid with less value (510.7) than the previous bid and see what happens.



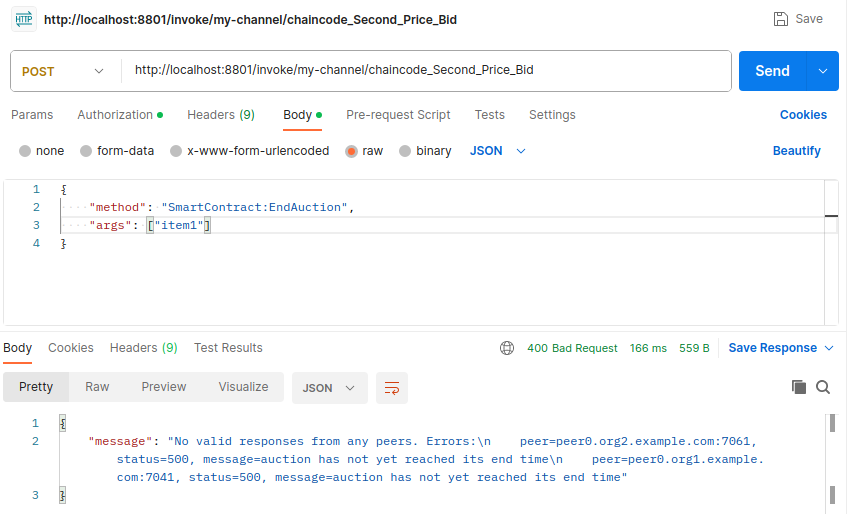
* Surprisingly the chaincode did not return any error. Now let’s bid for an item (“*item2*”) for which no auction has been started.



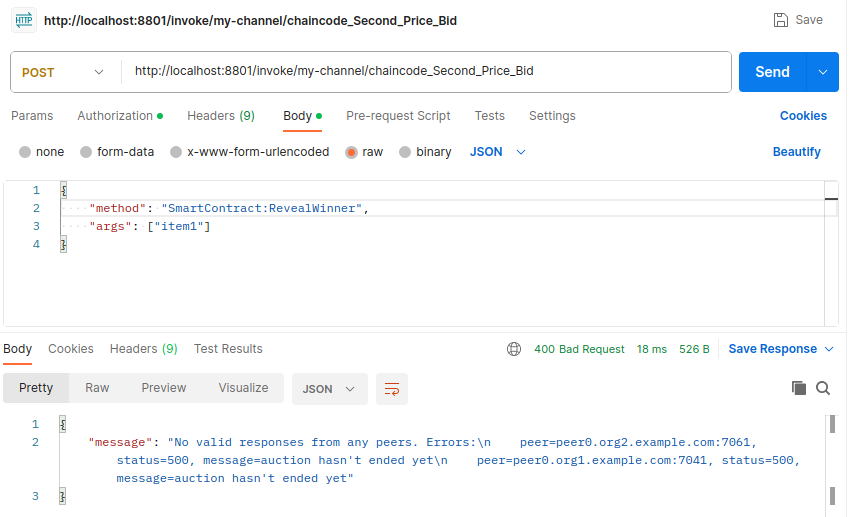
* Now we have received an error saying “*auction item not found*”. So, why is that? Why are we getting errors for one invalid case, but no error for another? Let’s see the PlaceBid method in the chaincode code.



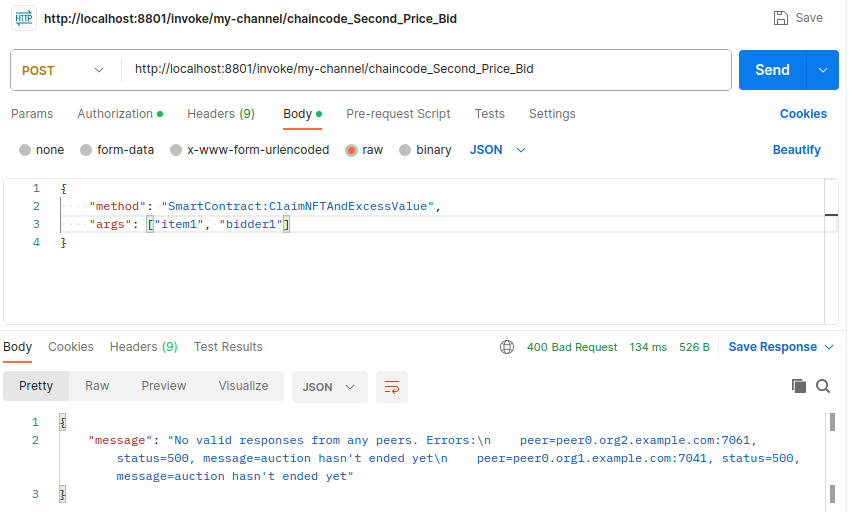
* We can see that the method is not complete. Many logics are missing from the function.
* Now let’s end the auction and reveal the winner by invoking the EndAuction and RevealWinner method.



* It is saying “*auction has not yet reached its end time*” as the end time was set to 22 November which has not passed yet.

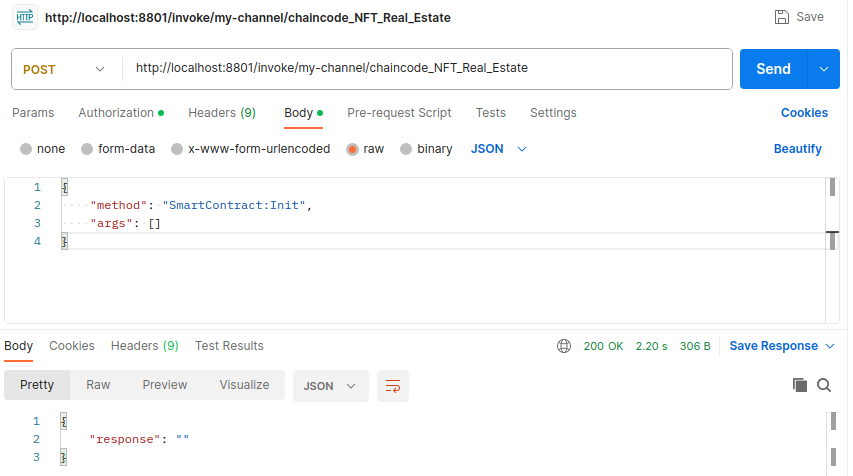


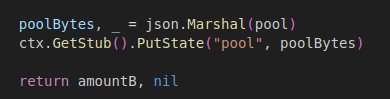
* Also, we can not reveal the winner as the auction can not be ended now. So, these methods are working perfectly. Now let’s invoke the last method of this chaincode named ClaimNFTAndExcessValue.



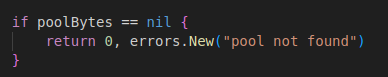
* We are again getting the same error saying “*auction hasn't ended yet*”. Perfect!

**Error Fixing**

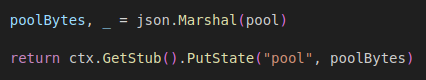
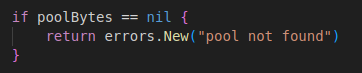
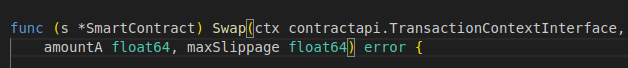
* We have completed testing all the 8 chaincodes. However, for 4 of those chaincodes, we were getting the following error while invoking any method.
* Error: endorsement failure during invoke. response: status:500 message:"error in simulation: failed to execute transaction<transaction\_number>: could not launch chaincode <chaincode\_name>.0.1:<transaction\_number>: chaincode registration failed: container exited with 2"
* After investigating all 4 chaincodes, we found some major similarities among 3 of those chaincodes (**First Price Bid**, **NFT Event Ticket**, **NFT Real Estate**), that are not present in the chaincodes that were tested okay. Some of them are as follows
  + *stub shim.ChaincodeStubInterface* instead of *ctx contractapi.TransactionContextInterface*
  + *return shim.Success(nil)* instead of *return nil*
  + *pb.Response* instead of *error*
  + *shim.Error* instead of *fmt.Errorf*
* Basically, all the chaincodes that generated the error used 2 common libraries from GitHub named [shim](http://github.com/hyperledger/fabric-chaincode-go/shim) and [peer](http://github.com/hyperledger/fabric-protos-go/peer). In the latest versions of Hyperledger Fabric, these 2 libraries are not used anymore. As our Fablo network runs the latest Hyperledger Fabric code, it can’t invoke the chaincodes that use those incompatible libraries.
* But, the good thing is we can modify the backdated chaincodes, and replace those 2 libraries with newer ones.
* We did this for the **NFT Real Estate** chaincode and it ran perfectly.
* All the codes will be found in this [GitHub repo](https://github.com/tanujsurve/Hyperledger_chaincode). The fixed chaincode\_NFT\_Real\_Estate chaincode can be found in [this commit](https://github.com/tanujsurve/Hyperledger_chaincode/commit/fae0c1891bd8da75ff755198d08fee96b829b72d). The previous code can be found in the [previous commit](https://github.com/tanujsurve/Hyperledger_chaincode/commit/20a392dcf7b82d43be3f7c76a155acc2e0ed15ff#diff-1f6cc1df4bdd81a469ba77945c615cd3d8140b9858e182957393a6c1cba5fa13)
* Now we have another chaincode **DEX AMM** that has some other issues.
  + Returning something else instead of the result from ctx.GetStub().PutState() method call.



* + Returning something else with an error object.





* Let’s fix this chaincode by changing these parts.
  + 
  + 
  + 
* After fixing those parts we ran the chaincode again. And it ran perfectly.