**Purchase Order Application** app is most important document in Supply Chain. This DAPP does demonstrates PO creation and state transitions of PO when buyer and seller performs respective actions in dapp. Its POC for creation and Workflow management of Purchase Order of supply chain in Decentralized world using IPFS and Ethereum.

**Backend:**

Following is backend workflow in dapp:

* Buyer creates the PurchaseOrder with all the details like PO number(should be unique, else replaces the existing one), Item Name, Item count, Seller, Unit Price, Incoterm, Destination. <To match real world use case, we should add more details like arrival date , discounts, line items as PO is extensive document >
* We hash the entire PO and store it in IPFS (Interplanetary File System P2P DB), as its very expensive and inefficient to store the whole document on Ethereum as PO undergoes many negotiations in real world use case.
* In Smart Contract we store, PONumber, IPFS hash, WorflowState, buyer and seller.
* Conceived PO Workflow States are CREATED, OPENED, CONFIRMED and DECLINED
* When PO is successfully created, it enters “CREATED” state.
* Buyer is responsible for approving as second step (In actual use case, there will be approval department in Buyer orgs)
* When PO is approved by Buyer, it enter “OPENED” state.
* Once PO is in “OPENED” state, its accessible by “Seller” now and locked for Buyer access.
* Seller can retrieve the PO information and can choose to “CONFIRM” or “DECLINE” the order.
* Accoridingly PO enters “CONFIRMED” or “DECLINED” state.
* The status of the order can be read by both buyer and seller at any given time.
* The contract owner can be buyer too.

**Front End UI:**

Front End UI is simple, unrefined, but made sure to demonstrate above described PO transitions. Its written in React.

* UI is split into 3 sections marked by horizontal line.

**First section**

* First section displays current ethereum account address from metamask or uport.

**Second section**

* Second section displays Purchase Order form accessible to Buyer Org (in real world use case), where PO details can be entered and submitted.
* Once buyer submits, in the backend, the data gets persisted in IPFS and Ethereum respectively.
* Second section also displays just created PO’s IPFS hash.

**Third Section**

* Contains components accessible by both buyer and seller.
* Contains query input field whose input parameter should be PO number
* On clicking “Show Order from IPFS & State from Ethereum”, tabular data will be generated which displays Workflow state, Order Details, Buyer and Seller, corresponding to input PONumber
* Workflowstate, buyer and seller are retrieved from Ethereum.
* Order Details are retrieved from IPFS.
* At any given time only one row of data corresponding to query PO Number is displayed.
* Below the tabular data, there are 3 buttons – “Approve” (only for Buyer of query PO) , Confirm and Decline (Only for Seller of query PO)
* Once the PO is in created state, Buyer and Seller can do their respective actions to move the PO(in query box) to respective state.
* Once the respective action is successfully performed, you can retrieve the data again by clicking “Show Order from IPFS & State from Ethereum” and check on the changed order states.
* Only designated buyer and seller can execute successful transactions and move the PO across states.

**Smart Contract & UI:**

* PuchaseOrder.sol does majority of above described functions and inherits Owned.sol and mortal.sol.
* For testing, TestPurchaseOrder.sol is written and uses ProxyContract.sol using ProxyDesign Pattern to test.
* **Web3.js** Current provider of web3(ganache, rinekby ,ropsten)
* **Ipfs.js.** Install ipfs-api and add ipfs configurations in **ipfs.js**. IPFS is set and read using IPFS api through Infura IPFS node.
* **Truffle.js**.Rinkeby and Ropsten deployment is done through Infura third party node. We do not have run ethereum client locally then.The configuration are in truffle.js
* **StoreMyValue.js** Stored the abi and contract address to create contract instance. This has to be done everytime new migration happens.
* **App.js** Simple UI designed with React

**Commands:**

* Bring up your metamask.
* Run Ganache (UI based)
* Change the setting in Ganache to point to Ganache (http://127.0.0.1:7545) and metamask should reflect the accounts and balances from Ganache
* Run “truffle compile”
* Run “truffle migrate”
* Once successful, copy the abi and contract address from “PurchaseOrder.json” to “StoreMyValue.js”
* Cd to client directory. “cd client”
* Run “npm run start”
* <http://localhost:3000> is where you will see your UI.
* Now you should be able to create, approve, view PO from buyer account and confirm, decline PO and view from seller account successfully.
* For subsequent migration and sompilation, please run “truffle migrate --reset --compile-all”

**Deploy in Rinkeby/Ropsten:**

* Do the configuration changes in truffle.js
* Change to Rinkeby/Ropsten network in metamask/uport and get corresponding Rinkeby/Ropsten ethers.
* Run the command =>
* “truffle migrate --network rinkeby --reset --compile-all”
* “truffle migrate --network ropsten --reset --compile-all”