The most significant computer science project which I have been a part of is based on Blockchain. I have had the opportunity to work on a live industrial project based on Blockchain technology and have done research in this technology for 6 months. I have completed this project with a team of five while working as an Intern for NITTTR Chandigarh, MHRD, Govt. of India.

As no proper documentation was available online for implementing private blockchain using Hyperledger Fabric, major time was spent doing research. I worked on developing the backend part in Blockchain with two more people working in the same domain.

Initially, it was a bit challenging for me to write Smart Contracts in Golang. Smart Contract is the main component behind implementing the logical capabilities of the Blockchain network. It required a lot of troubleshooting for writing my first Smart Contract, and the most displeasing thing was to make the network down and then up every time any change had been made into the Smart Contract.
But after studying several types of available Smart Contracts and practicing them for 2 months, I became confident in writing got comfortable in writing Smart Contracts for any Blockchain network.

To store the data involved in a distributed manner for providing it utmost security, we used IPFS. The integration part was the toughest of the whole implementation, we had to integrate Backend consisting of Hyperledger Fabric, IPFS, the frontend consisting of the whole web app made in PHP and Apache Server used to provide security to the project. Each of my team members worked hard for making this project a major success.

Through this work, I have gained substantial knowledge of Enterprise Blockchain technology and along with that, I have learned the importance of teamwork in any project.

PROJECT -
HOSTEL ALLOTMENT SYSTEM USING HYPERLEDGER FABRIC

In Hyperledger, transactions are submitted via an interface to the ordering service. This service collects transactions based on the consensus algorithm and configuration policy, which may define a time limit or specify the number of transactions allowed. Most of the time, for efficiency reasons, instead of outputting individual transactions, the ordering service will group multiple transactions into a single block. In this case, the ordering service must impose and convey a deterministic ordering of the transactions within each block.

Advantages of using a Hyperledger based application:

- Private Blockchain hence student records are not in public domain.
- Access levels can be customized as per requirement. Students won't be able to allocate rooms to themselves.
- Beneficiary will only see the details of the diploma once it's been issued.
- As Hyperledger is not coin ("token") based blockchain, the environment is less complex to develop.
- Unlike Bitcoin or Ethereum blockchain Hyperledger does not require transferring a virtual currency to publish a transaction.
- You can query Hyperledger blockchain to extract details of students to whom hostel has been alloted.

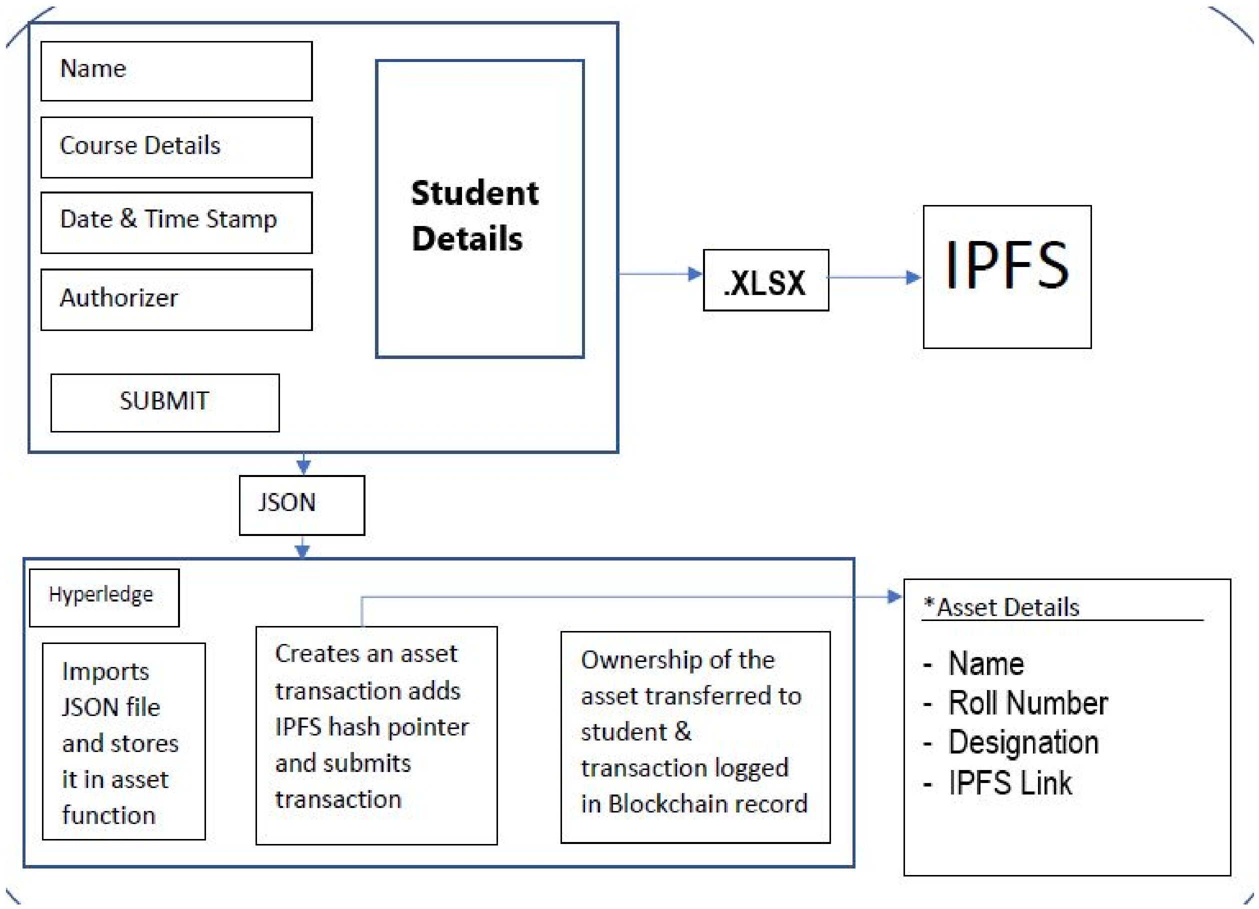
How it Works -

The application has three main components-
1) Front End.
2) Backend :-
a) Interplanetary File System (“IPFS”).
b) Hyperledger Fabric.

One of the disadvantages of storing information on a blockchain based application is that you cannot store files, so we overcame this issue by integrating our system with IPFS. IPFS is a peer-to-peer method of storing and sharing media in a distributed file system. It is based on Bitcoin blockchain protocol and stores unalterable data, removes duplicates across the network.

Student details such as Student ID, Student Name, Course Name, and Year of Graduation are added on the web application. The web application then generates an excel file containing all the details of student. The .xlsx file is uploaded to IPFS node and the hash-pointer (“IPFS link”) is saved on our blockchain. The JSON file (containing minimal details) is imported by Hyperledger to create an asset and an Invoke Event is displayed on the Blockchain Terminal.

Furthermore, as the system is hosted on Hyperledger blockchain we can efficiently manage the access levels. The beneficiary i.e. the institution which saves the details of the student as a proof of credentials will have a clear view of the history of the candidate. The .xlsx file will contain immutable record of the allotment done by the hostel authority, date and time of allotment and other details.


Name
Course Details
Date & Time Stamp
Authorizer
SUBMIT
Student Details
.XLSX
IPFS
JSON
Hyperledge
Imports JSON file and stores it in asset function
Creates an asset transaction adds IPFS hash pointer and submits transaction
Ownership of the asset transferred to student & transaction logged in Blockchain record
*Asset Details
- Name
- Roll Number
- Designation
- IPFS Link

Application Architecture

Now that we have an overview of how the application works let us take a closer look at implementation of each module.

InterPlanetary File System (IPFS):

IPFS is a peer-to-peer distributed file system that seeks to connect all computing devices with the same system of files. In few ways IPFS is similar to World Wide Web, but IPFS could be seen as a single BitTorrent swarm exchanging objects within one Git repository. IPFS took advantage of Bitcoin blockchain protocol and network infrastructure to store, unalterable data, remove duplicate files across the network and obtain address information for accessing storage nodes. IPFS has no single point of failure and nodes do not need to trust each other except for every node

they are connected to. Distributed Content Delivery saves bandwidth and prevents DDos attacks, which HTTP struggles with.
In-order to store information on IPFS network we first have to create a node, this can be achieved by downloading the IPFS infrastructure from https://ipfs.io/docs/install/. Once you have downloaded the files and have spun a node on your system we can run an IPFS daemon to connect to the Global Object Repository of IPFS.
IPFS returns hash values of the documents uploaded from any node. This hash value is also the location pointer to each document. We have to add the stem of the HTTPS protocol followed by the hash of the file. This link is then stored in the asset class of Hyperledger and can be utilized to view the actual file of the participant.

Hyperledger:

Hyperledger is a Linux foundation project to produce an open blockchain platform that is ready for business. It provides implementation of shared ledger, smart contracts, privacy and consensus mechanisms.

Process Flow :

In our model follows the following structure:
Participants:
a) Warden -The warden of each hostel in the university network.
b) Head of Department (HOD) - Head of the department in the university where student is currently enrolled.
c) Clerk - Clerk of the each hostel issuing fee receipt to the students on successful payment of hostel fees.
d) Student - Student of the university that wants hostel to be allocated to him/her.
Asset:
Person - Holds the person(Student, Warden, Clerk, HOD) details.
Transactions:

a) Invoke — Executed when any person in the university registers himself/herself in the university network and when any change is made to the details.
b) Query — Executed when any person requests to see the details to check the current status of the allocation process.

<u>A quick walkthrough of the project:</u>
1. First of all, participants will register themselves in the web app to become a part of the network. In the backend, invoke function will be called to make a new entry in the blockchain Ledger (Refer to the sample code). There would be different predefined formats for registration IDs of authority members and students.
2. If a student registers himself/herself in the network, the request will be sent to the warden of the respective hostel to check the hostel availability for the student.
3. After the verification done by the warden, the request will be sent to the head of the department (HOD) to verify the student enrollment in the department.
4. When the HOD verifies the request, further request will be sent to the clerk of the respective hostel who will issue a fee receipt to the student after successful payment of the hostel fees.
5. After the verification done by the clerk, the request will be sent to the warden of the respective hostel to allocate the available room to the student.
6. Any change in the participant's form will lead to a separate transaction in Blockchain Ledger recorded along with the time stamp. In the backend, **invoke** function will be called if any change is made in the file (Refer to the sample code). This leads to the immutability of the Blockchain Ledger.
7. If the authority member wants to check the history of a student, he/she can query through the blockchain using the student's registration number. In the backend, **getHistory** function will be called to fetch the full history of a student. (Refer to the sample code)

8. If the warden wants to check the list of all the residents of the respective hostel, he/she can query through the blockchain. In the backend, **getall** function will be called to fetch the list of all the students residing in the respective hostel (Refer to the sample code).
9. To ensure security of the data involved in the model, IPFS is used. IPFS will store all the files containing details of the participants in a distributed manner. Stored data is safe even if the main server gets hacked. The file will come to the server's existence only if any query is made through blockchain to check the participant's details. For instance, If there's a need to check the current status of a student's hostel allotment process. As soon as, an exit is made from the student details being shown, the file will get deleted from the server. In the backend, to implement this functionality a script has been written in Golang.
10. The connectivity of IPFS with the Blockchain provides the utmost security to the whole project model.

Sample Code -

Smart Contract used in the project -

```
package main

import (
        "fmt"
        "time"
        "bytes"
        "strings"
        "strconv"
        "encoding/json"
        "github.com/hyperledger/fabric/core/chaincode/shim"
        "github.com/hyperledger/fabric/protos/peer"
)

// SimpleAsset implements a simple chaincode to manage an asset
```

```
type SmartContract struct {
}
type Person struct {
 Name string `json:"name"`
 Rollno int `json:"rollno"`
 Designation string `json:"desg"`
 Hash string `json:"hash"`
}

// Init is called during chaincode instantiation to initialize any
// data. Note that chaincode upgrade also calls this function to reset
// or to migrate data.
func (t *SmartContract) Init(stub shim.ChaincodeStubInterface) peer.Response {
	// Get the args from the transaction proposal
	args := stub.GetStringArgs()
	var check bool
	if len(args) != 5 {
	return shim.Error("Incorrect arguments, Expecting 5")
	}

	if len(args[4])!= 46 {
	 return shim.Error(fmt.Sprintf("hash should be 46 characters long"))
	}
	check = strings.HasPrefix(args[4], "Qm")
	if check != true {
	 return shim.Error(fmt.Sprintf("hash should start with Qm"))
	}
	rollno,errp := strconv.Atoi(args[2])
	if errp != nil {
	 return shim.Error(fmt.Sprintf("Error starting SmartContract chaincode: %s", errp))
	}
	var  data  =  Person{Name:  args[1],  Rollno:  rollno,  Designation:  args[3],  Hash:
args[4]}
	PersonBytes, _ := json.Marshal(data)
	err := stub.PutState(args[2], PersonBytes)
	if err != nil {
	return shim.Error(fmt.Sprintf("Failed to create record for: %s", args[2]))
	}
	return shim.Success(nil)
```

```
}
// Invoke is called per transaction on the chaincode. Each transaction is
// either a 'get' or a 'set' on the asset created by Init function. The Set
// method may create a new asset by specifying a new key-value pair.
func (t *SmartContract) Invoke(stub shim.ChaincodeStubInterface) peer.Response {
        // Extract the function and args from the transaction proposal
        fn, args := stub.GetFunctionAndParameters()

        var result string
        var err error
        if fn != "invoke" {
        return shim.Error("unknown fxn call")

        }
        if args[0] == "set"{
                result,err = set(stub, args)
        } else if args[0] == "clerk" || args[0] == "hod"{ // assume 'get' even if fn is nil
        result, err = get(stub, args)
        } else if args[0] == "getHistory"{
         return getHistory(stub, args)
        } else if args[0] == "accounts"{
         return getall(stub)
        } else if args[0] == "warden"{
         if args[1] == ""{
                 return getall(stub)
         } else {
                 result, err = get(stub, args)
         }
        }
        if err != nil {
        return shim.Error(err.Error())
        }
        return shim.Success([]byte(result))

}

// Set stores the asset (both key and value) on the ledger. If the key exists,
// it will override the value with the new one
func set(stub shim.ChaincodeStubInterface, args []string) (string, error) {
        var check bool
        if len(args) != 5 {
        return "", fmt.Errorf("Incorrect arguments, Expecting 5")
        }
        if len(args[4])!=46 {
        return "", fmt.Errorf("hash should be 46 characters long")
        }
        check = strings.HasPrefix(args[4], "Qm")
        if check != true {
         return "", fmt.Errorf("hash should start with Qm")
        }
        rollno, errp := strconv.Atoi(args[2])
        if errp != nil {
         return "",fmt.Errorf("Error starting SmartContract chaincode: %s", errp)
        }
        var  data  =  Person{Name:  args[1],  Rollno:  rollno,  Designation:  args[3],  Hash:
args[4]}
        PersontBytes, _ := json.Marshal(data)
        err := stub.PutState(args[2], PersonBytes)
        if err != nil {
        return "", fmt.Errorf("Failed to set asset: %s", args[2])
        }
        // Notify listeners that an event "eventInvoke" have been executed (check line 19
in the file invoke.go)
        err = stub.SetEvent("eventInvoke", []byte{})
        if err != nil {
        return "", fmt.Errorf("Failed to set asset: %s", args[2])
}
        return args[2], nil
}

// Get returns the value of the specified asset key
func get(stub shim.ChaincodeStubInterface, args []string) (string, error) {
        if len(args) != 2 {
        return "", fmt.Errorf("Incorrect arguments. Expecting a key")
        }

        PersonBytes, err := stub.GetState(args[1])
        if err != nil {
```

```
        return "", fmt.Errorf("Failed to get asset: %s with error: %s", args[1], err)
        }
        if PersomBytes == nil {
        return "", fmt.Errorf("Asset not found: %s", args[1])
        }
        return string(PersonBytes), nil
}
func getHistory(stub shim.ChaincodeStubInterface, args []string) peer.Response {
    if len(args) < 2 {
            return shim.Error("Incorrect number of arguments, Expecting 2")
    }
    PersonId := args[1]
    fmt.Printf("- start getHistory: %s\n", PersonId)
    resultsIterator, err := stub.GetHistoryForKey(PersonId)
    if err != nil {
            return shim.Error(err.Error())
    }
    defer resultsIterator.Close()

    //buffer is a json array containing historic values for the person
    var buffer bytes.Buffer
    buffer.WriteString("[")

    bArrayMemberAlreadyWritten := false
    for resultsIterator.HasNext() {
            response, err := resultsIterator.Next()
            if err != nil {
                    return shim.Error(err.Error())
            }
            //Add a comma before array members, suppress it for the first array member
    if bArrayMemberAlreadyWritten == true {
            buffer.WriteString(",")
    }
    buffer.WriteString("{\"TxId\":")
    buffer.WriteString("\"")
    buffer.WriteString(response.TxId)
    buffer.WriteString("\"")
    buffer.WriteString(", \"Value\":")
    //if it was a delete operation on given key, then we need to set the
    //corresponding value null.Else, we'll write the response.Value
    //as-is (as the Value itself a JSON person)
    if response.IsDelete {
            buffer.WriteString("null")
    } else {
            buffer.WriteString(string(response.Value))
    }
    buffer.WriteString(",\"Timestamp\":")
    buffer.WriteString("\"")
                    buffer.WriteString(time.Unix(response.Timestamp.Seconds+19800,     int64
(response.Timestamp.Nanos)).String())
    buffer.WriteString("\"")

           buffer.WriteString("}")
           bArrayMemberAlreadyWritten = true
    }
    buffer.WriteString("]")
    fmt.Printf("- getHistory returning:\n%s\n", buffer.String())
           fmt.Printf("\n");
    return shim.Success(buffer.Bytes())
}
func  getall(stub shim.ChaincodeStubInterface) peer.Response {
           startkey := "172000"
           endkey := "172100"

           PersonBytes, err := stub.GetStateByRange(startkey,endkey)
           if err != nil {
            return shim.Error(err.Error())
    }
           defer PersonBytes.Close()
           var buffer bytes.Buffer
           for PersonBytes.HasNext() {
           queryResponse, err := PersonBytes.Next()

           if err != nil {
           return shim.Error(err.Error())
           }
           buffer.WriteString(string(queryResponse.Value))
           }
```

```
        fmt.Printf("-all query:\n%s\n",buffer.String())
        return shim.Success(buffer.Bytes())
}
// main function starts up the chaincode in the container during instantiate
func main() {
        err := shim.Start(new(SmartContract));
        if  err != nil {
        fmt.Printf("Error starting SmartContract chaincode: %s", err)
        }
}
```

<u>Code used to for transactions in the project</u> -

a) For making query in the blockchain:-

```
package blockchain

import (
    "fmt"
    "github.com/hyperledger/fabric-sdk-go/pkg/client/channel"
)

// QueryHello query the chaincode to get the state of hello
func (setup *FabricSetup) QueryHello(value []string) (string, error) {
    if len(value) != 2 {
         fmt.Printf("Incorrect no of arguments")}

    var args []string
    args = append(args, "invoke")
    args = append(args, value[0])
        args = append(args, value[1])
    response, err := setup.client.Query(channel.Request{ChaincodeID:
setup.ChainCodeID,            Fcn:            args[0],            Args:
[][]byte{[]byte(args[1]),[]byte(args[2])}})
    if err != nil {
         return "", fmt.Errorf("failed to query: %v", err)
    }

    return string(response.Payload), nil
}
```

b) For making invoke in the blockchain:-

```
package blockchain

import (
    "fmt"
    "github.com/hyperledger/fabric-sdk-go/pkg/client/channel"
    "time"
)

// InvokeHello
func (setup *FabricSetup) InvokeHello(args []string) (string, error) {
    //if len(args) != 4 {
        //fmt.Printf("incorrect no of arguments")}

    eventID := "eventInvoke"

    // Add data that will be visible in the proposal, like a description of
the invoke request
    transientDataMap := make(map[string][]byte)
    transientDataMap["result"] = []byte("Transient data in hello invoke")

                                        reg,        notifier,        err        :=
setup.event.RegisterChaincodeEvent(setup.ChainCodeID, eventID)
    if err != nil {
        return "", err
    }
    defer setup.event.Unregister(reg)
    // Create a request (proposal) and send it
    response, err := setup.client.Execute(channel.Request{ChaincodeID: setup.ChainCodeID, Fcn: "invoke", Args: [][]byte{[]byte("set"), []byte(args[0]), []byte(args[1]), []byte(args[2]),[]byte(args[3])}, TransientMap: transientDataMap})
    if err != nil {
        return "", fmt.Errorf("failed to move funds: %v", err)
    }

    // Wait for the result of the submission
    select {
    case ccEvent := <-notifier:
    fmt.Printf("Received CC event: %s\n", ccEvent)
    case <-time.After(time.Second * 20):
        return "", fmt.Errorf("did NOT receive CC event for eventId(%s)", eventID)
    }

    return string(response.TransactionID), nil
}
```

System Requirements -

- This application has been made on Ubuntu 16.04 using Hyperledger Fabric framework and GoLang.
- Hyperledger Fabric is a platform for distributed ledger solutions underpinned by a modular architecture delivering high degrees of confidentiality, resiliency, flexibility and scalability. It is designed to support pluggable implementations of different components and accommodate the complexity and intricacies that exist across the economic ecosystem.

Technology Stack Used -

- *Hyperledger Fabric*
- *Golang*
- *IPFS*
- *PHP*
- *Apache Server*
- *Mongodb*

Conclusion:

During the recent few months blockchain based enterprise solution has been a topic of prime interest. A hostel allotment system based on blockchain is one of the key use case of Blockchain. Using Blockchain, it is now possible for us to store our data in a secure way.

A similar infrastructure could be designed which could handle the transfer of medical documents. Prescriptions can be created by doctors; each prescription can be added as an asset in the Hyperledger application and file can be uploaded on IPFS. The patient can then show the prescription to any chemist and the chemist can verify the authenticity of the prescription with the history of the doctor who issued it. Such a system can further evolve in an established Healthcare Records Management System.