Indian Institute of Technology Kanpur

Department of Computer Science and Engineering

CS731 - Blockchain and its Applications

End Semester Project Report

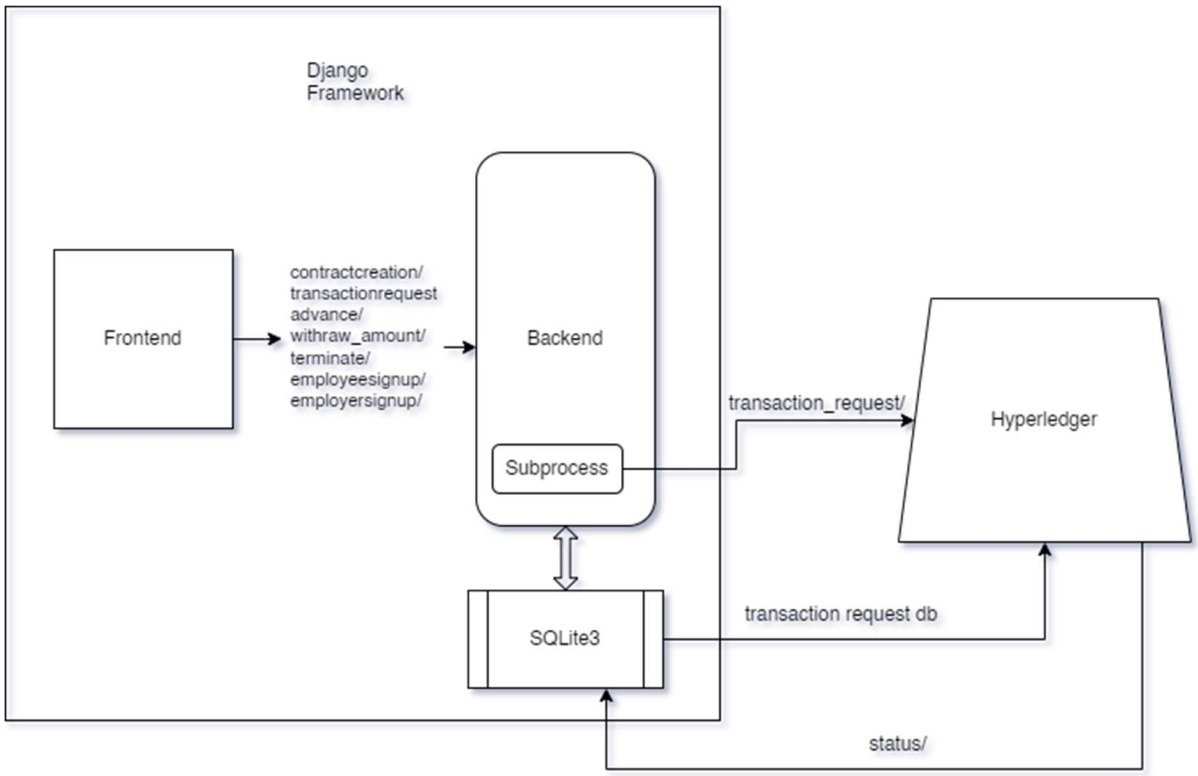
BorderPay.io - Hyperledger based Payroll System for Cross-Border Transactions

Submitted By

S.No.	**Name**	**Roll No.**
1	Saksham Arora	200843
1	Aryan Bansal	200198

Contents

Architecture



Introduction

BorderPay.io is a complete system that allows for the creation of financial contracts between a company and its contractors/employees, with a progressive payroll system that automatically updates and calculates salaries of the contractors/employees on a timely basis. It ensures payments are made as per the contract signed by both parties, whether they are local or cross-border.

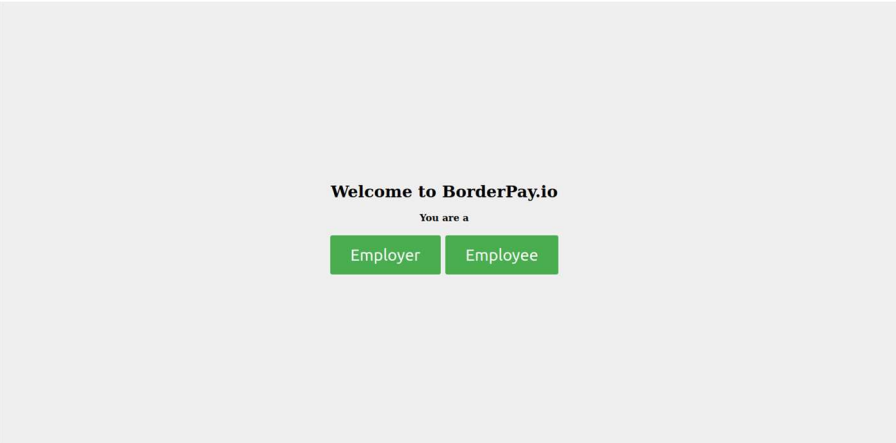
Implementation

Frontend

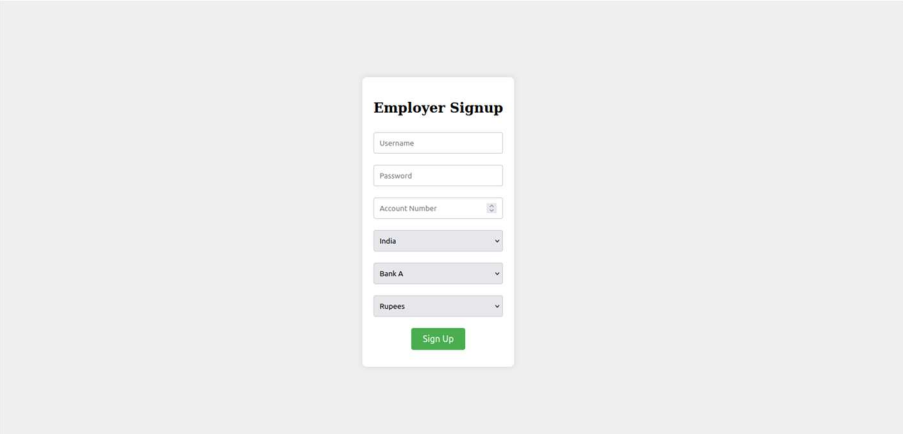
Tech Stack used:

- HTML
- Bootstrap CSS

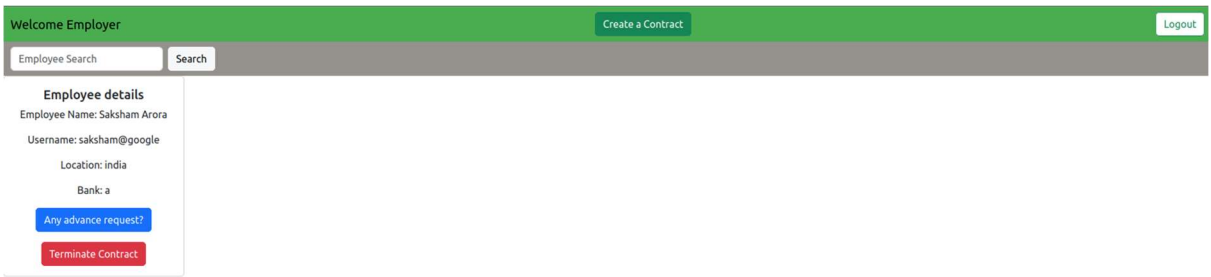
Below are the wireframes of our website:



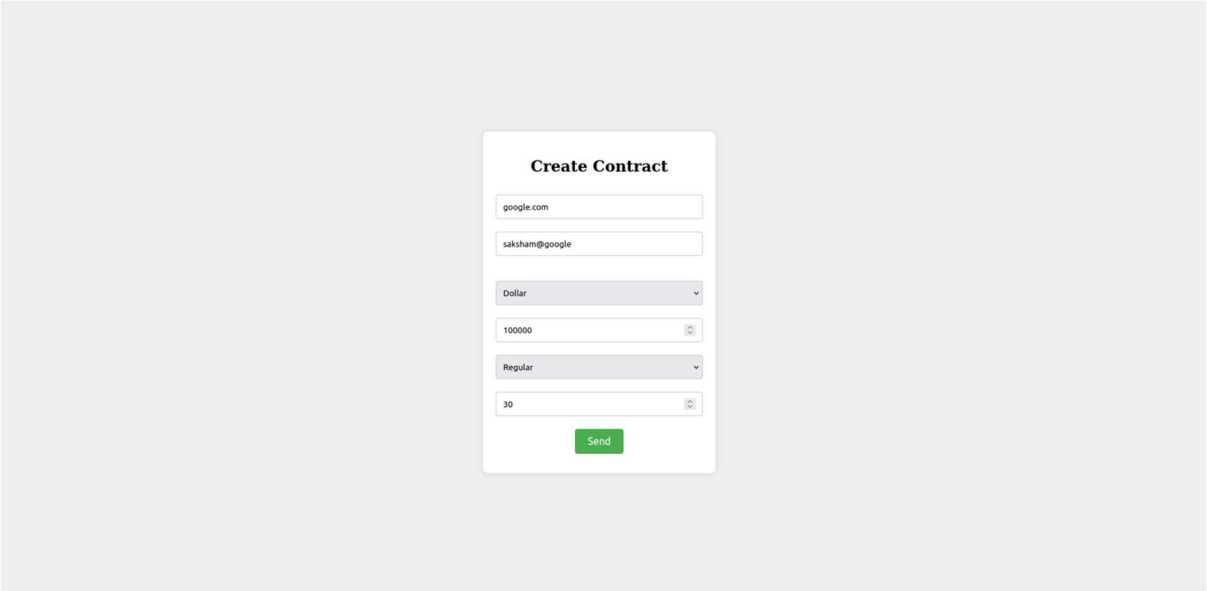
Landing Page



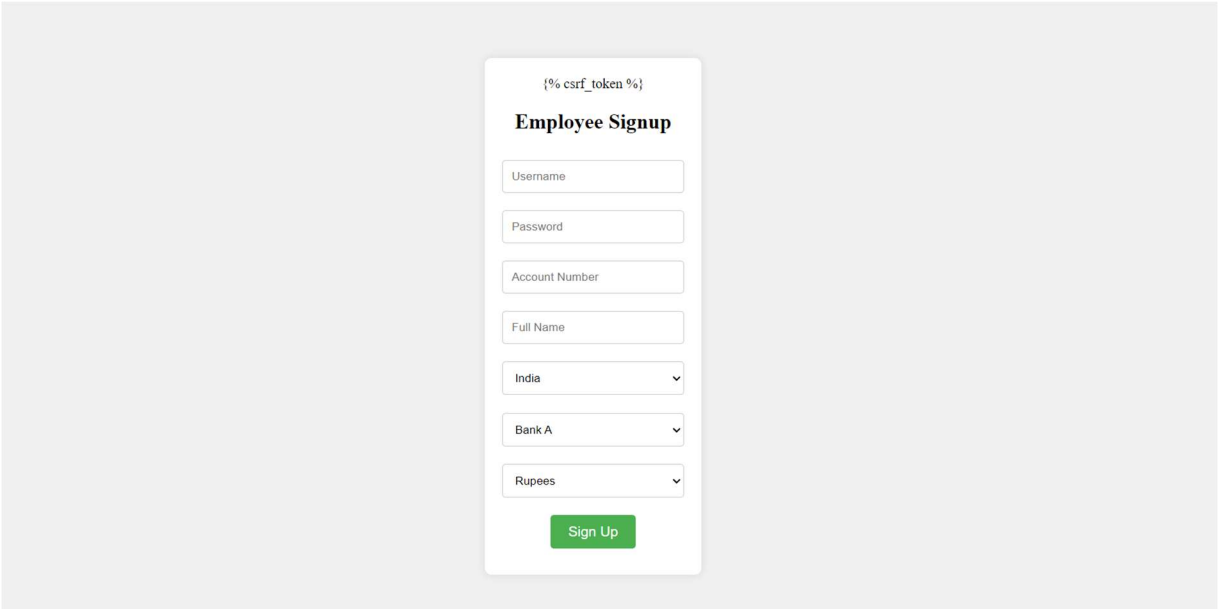
Signup page for Employer



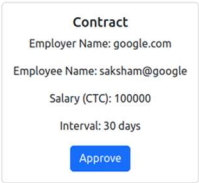
Employer Dashboard



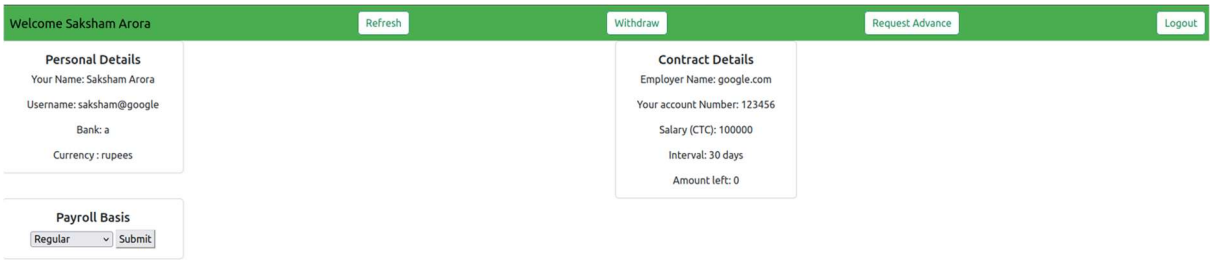
Create contract



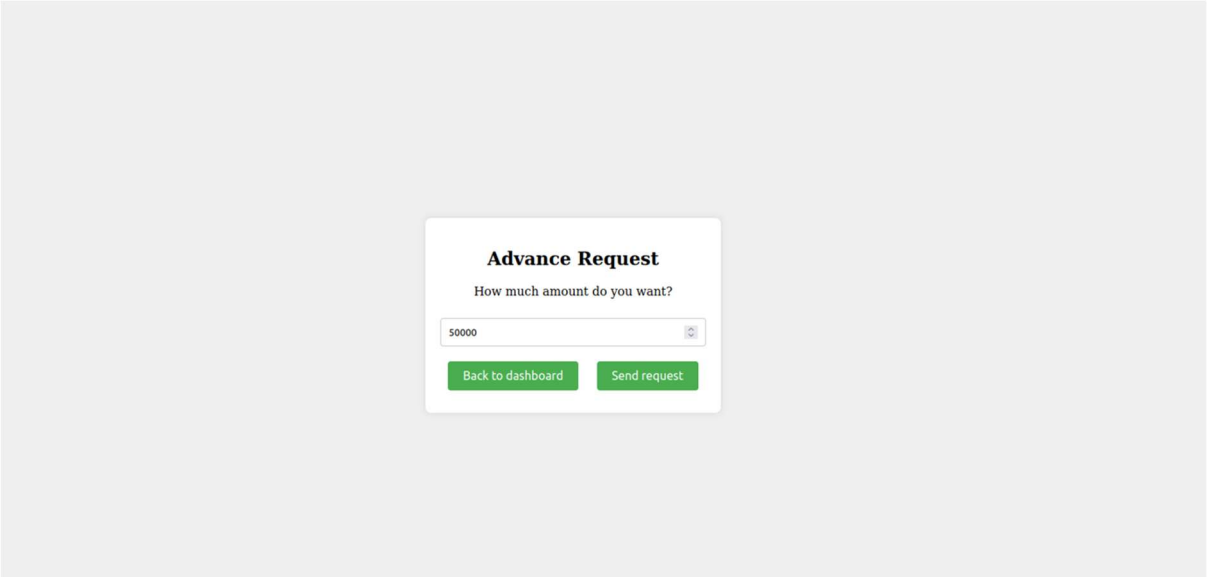
Employee Signup



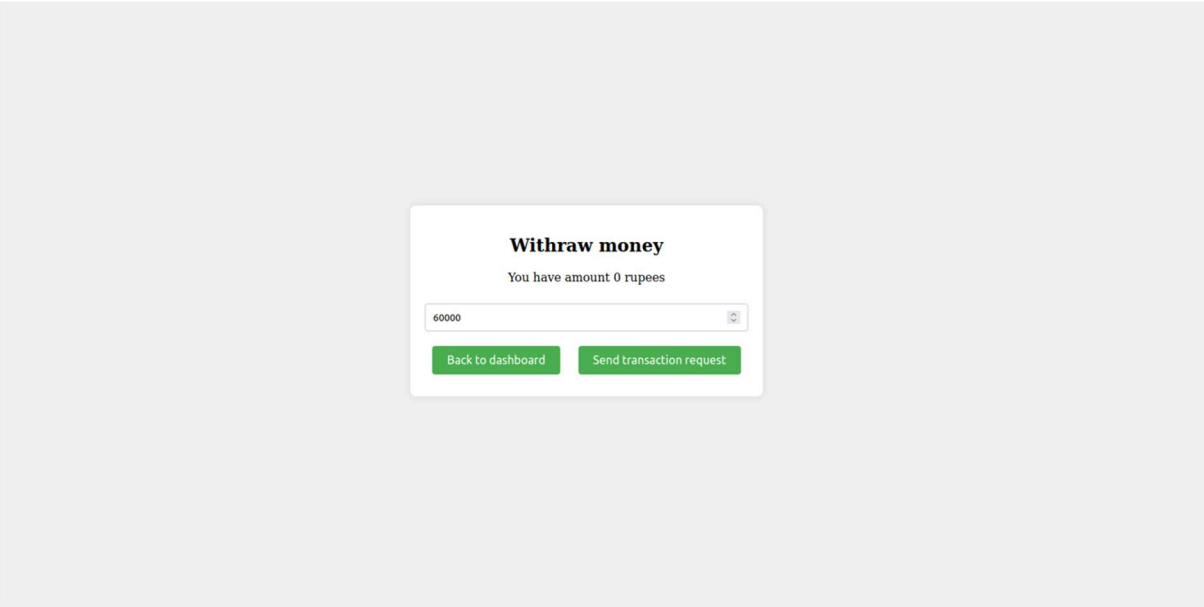
Employee: Contract Approval



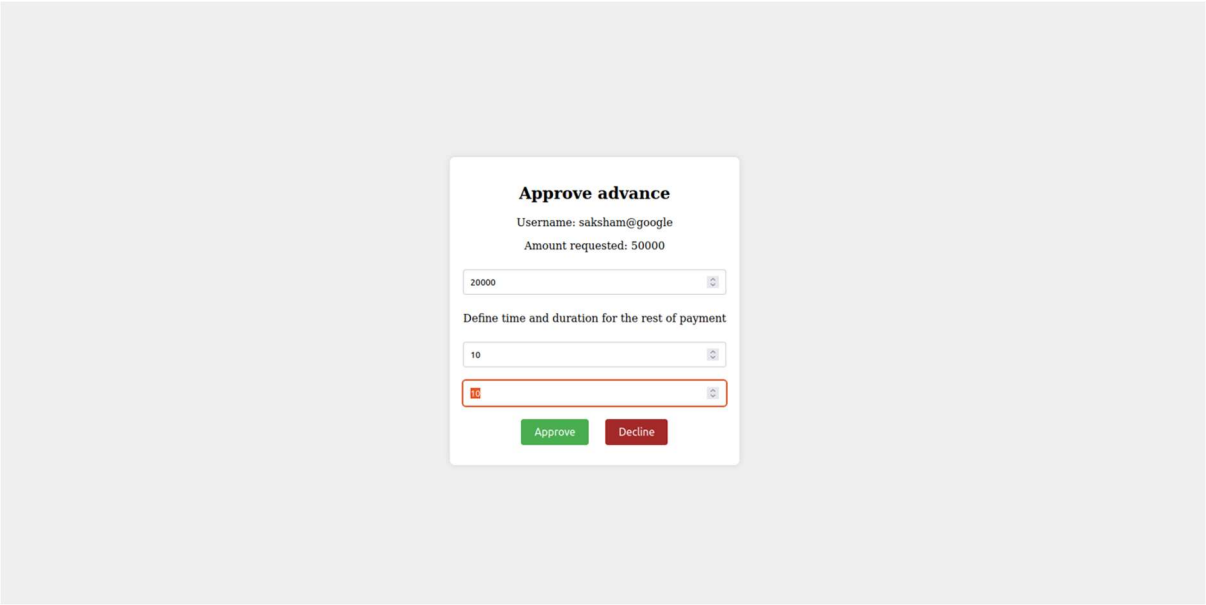
Employee Dashboard



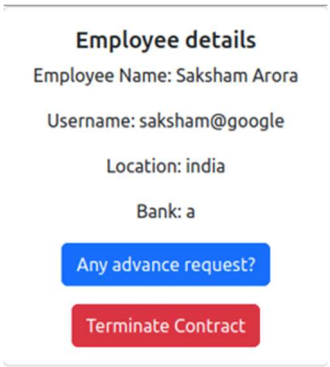
Employee: Initiate Advance Request



Employee: Withdraw Money



Employer Advance Approval



Employee details on Employer dashboard

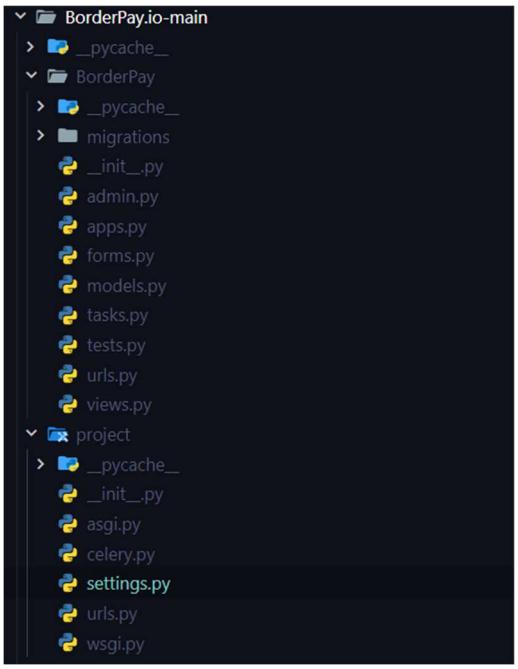
Backend

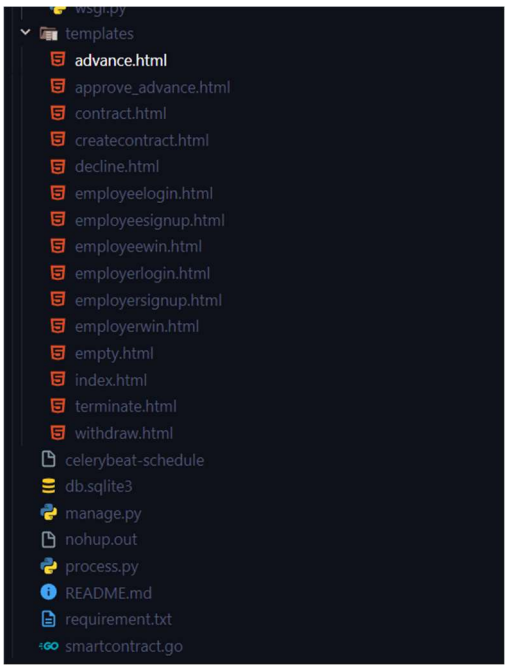
Tech Stack Used:

- Django
- Django – SQLite3 Database

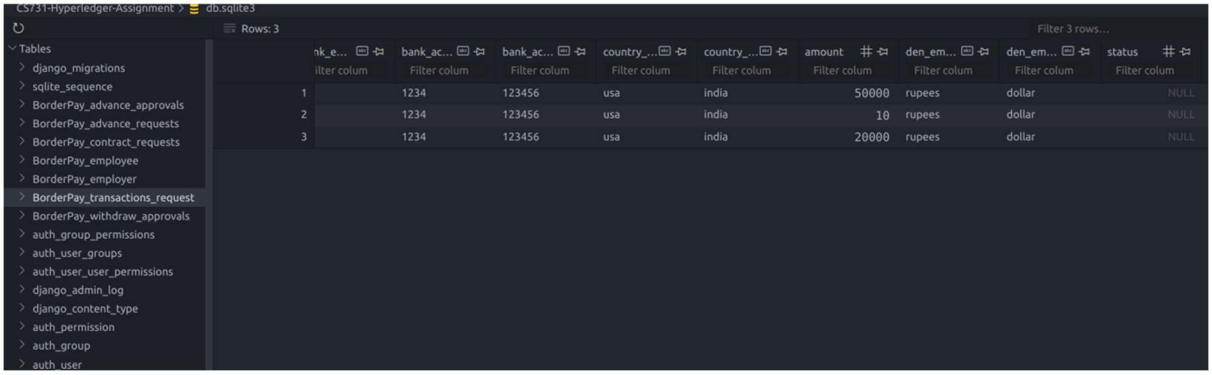
Core functionality:

- It is where we have done all the interaction with the Database and then further with the Hyperledger.
- Basic functionality includes, **Contract Agreement, Automatic Interval based Salary, 2 types of Payroll basis, Contract termination, Advance Requests, Withdrawal Requests, Cross-Border Transactions.**
- We have used Django which provides a complete package for fronted and backend. The workflow happens on the frontend thereby updating the database on a regular basis.
- We have employed **Celery** to implement a time-based scenario which automatically transfers the salary, after every set interval.
- We have tried to integrate the Backend with Hyperledger using Python Subprocess commands, which can be found in the GitHub Repository.
- Following are the snapshots from our backend directory structure:





- The users interact with the UI, meanwhile Django keeps updating the database. Each time a **Transaction Request** is generated, the *Transaction Request* database updates and a API call is made using Python subprocess to initiate the transaction on the **Hyperledger.**



After a successful transaction from the Hyperledger, the *status* gets updated to *successful.*

Hyperledger

Tech Stack used:

- GoLang
- Python Subprocess

We have implemented our **chaincodes** using **Golang.**
The code involves taking input from the database and perform a transaction on the Hyperledger, whilst updating the ledger and updating the Django database too.

Here or some screenshots from the chaincode:

```
assets := []Asset{
    {Account: "23456", Name: "Optiver", Balance: 100000000, Bank: "b", Denom: "dollar", Loca: "USA"},
    {Account: "34567", Name: "Google", Balance: 100000000, Bank: "c", Denom: "euro", Loca: "Netherlands"},
    {Account: "45678", Name: "Oracle", Balance: 100000000, Bank: "a", Denom: "dollar", Loca: "Canada"},
    {Account: "56789", Name: "EXL", Balance: 100000000, Bank: "b", Denom: "dollar", Loca: "Australia"},
    {Account: "67890", Name: "Aryan", Balance: 100, Bank: "c", Denom: "rupees", Loca: "India"},
    {Account: "78901", Name: "Saksham", Balance: 100, Bank: "a", Denom: "dollar", Loca: "USA"},
    {Account: "89012", Name: "Shubham", Balance: 100, Bank: "b", Denom: "euro", Loca: "Netherlands"},
    {Account: "90123", Name: "Sachin", Balance: 100, Bank: "c", Denom: "dollar", Loca: "Canada"},
    {Account: "12345", Name: "Ujwal", Balance: 100, Bank: "a", Denom: "dollar", Loca: "Australia"},
}
```

This is what we have initially given data to the ledger.

```go
const (
    rupeeToDollar = 0.014
    rupeeToEuro   = 0.012
    dollarToRupee = 72.0
    dollarToEuro  = 0.85
    euroToRupee   = 88.5
    euroToDollar  = 1.18
)
```

Forex rates to deal with Cross border payments.

```go
type Asset struct {
    Account        string    `json:"Account"`
    Name          string `json:"Name"`
    Balance        int `json:"Balance"`
    Bank          string `json:"Bank"`
    Denom          string    `json:"Denom"`
    Loca        string `json:"Loca"`
}
```

The asset object to deal with the Transactions.

```
saksham2048@sakshams-pc: ~/go/src/github.com/sakshamar...
RootCertFiles "${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2
.example.com/tls/ca.crt" -c '{"function":"TransferAsset","Args":["23456","67890", "1000
0"]}'
2024-05-01 23:56:28.468 IST 0001 INFO [chaincodeCmd] chaincodeInvokeOrQuery -> Chaincod
e invoke successful. result: status:200 payload:"true"
saksham2048@sakshams-pc:~/go/src/github.com/sakshamar20/fabric-samples/test-network$ pe
er chaincode query -C mychannel -n basic -c '{"Args":["GetAllAssets"]}'
[{"Account":"12345","Name":"Ujwal","Balance":100,"Bank":"a","Denom":"rupees","Loca":""}
,{"Account":"23456","Name":"Optiver","Balance":99979900,"Bank":"b","Denom":"dollars","L
oca":""},{"Account":"34567","Name":"Google","Balance":100000000,"Bank":"c","Denom":"eur
o","Loca":""},{"Account":"45678","Name":"Oracle","Balance":100000000,"Bank":"a","Denom"
:"rupees","Loca":""},{"Account":"56789","Name":"EXL","Balance":100000000,"Bank":"b","De
nom":"dollar","Loca":""},{"Account":"67890","Name":"Aryan","Balance":20100,"Bank":"c","
Denom":"euro","Loca":""},{"Account":"78901","Name":"Saksham","Balance":100,"Bank":"a","
Denom":"rupees","Loca":""},{"Account":"89012","Name":"Shubham","Balance":100,"Bank":"b"
,"Denom":"dollar","Loca":""},{"Account":"90123","Name":"Sachin","Balance":100,"Bank":"c
","Denom":"euro","Loca":""}]
saksham2048@sakshams-pc:~/go/src/github.com/sakshamar20/fabric-samples/test-network$
```

We initially ran the Hyperledger using script, which fetches out the ledger results each time a transaction is initiated.