BorderPay – Contractual Payments

Ahmad Amaan 200062
Himanshu Jindal 200442

Introduction: BorderPay is used for managing users and contracts, and facilitating financial transactions like salary payments between employers and employees.
The primary objective is to manage users, handle contracts, and process transactions within a secure blockchain environment. Some other functionalities include user registration, contract creation, salary transactions, and data retrieval.

Frontend:

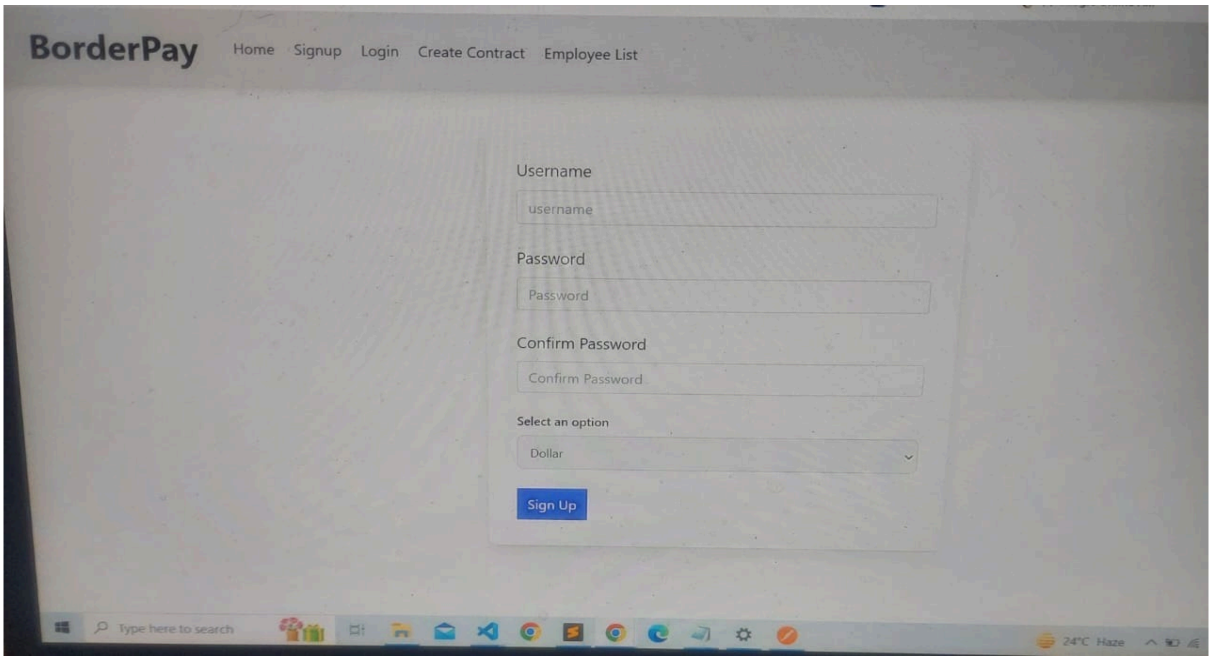
Technologies used:

- HTML
- Tailwind CSS
- JavaScript
- React
- Yup

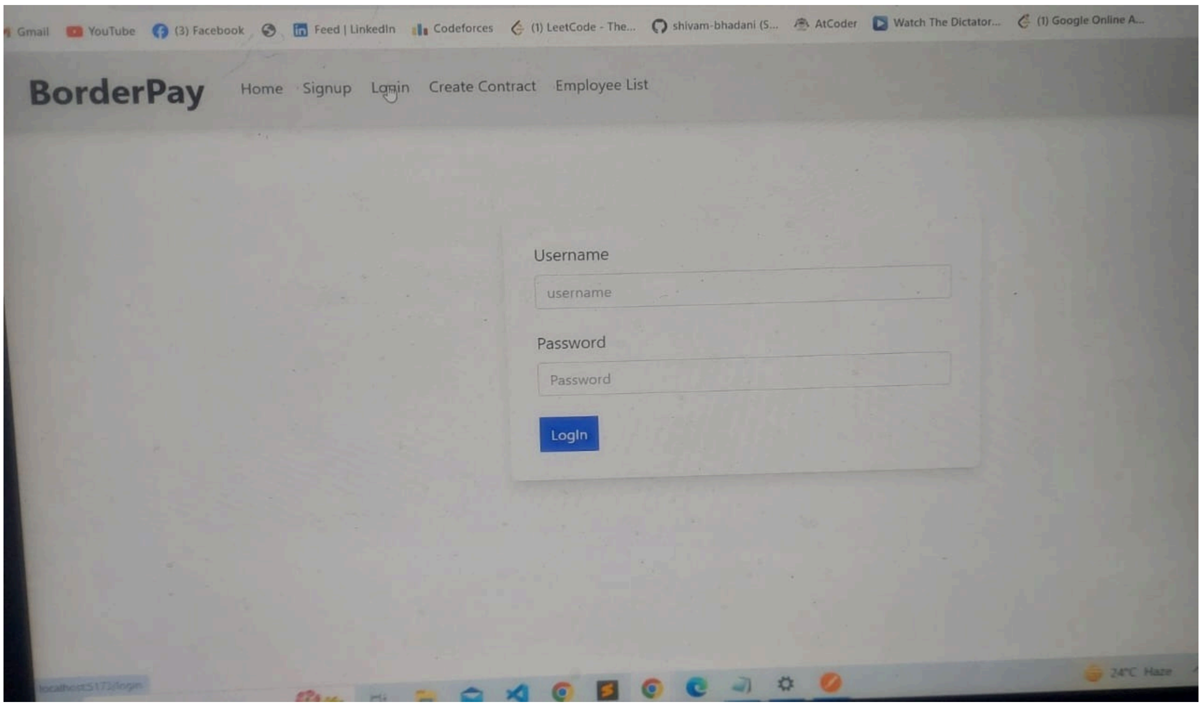
Major Components of Frontend are:

- **Signup/login page -** User can signup as employer or employee. Already signed in users can login to their dashboard. Yup library is used for form validation .

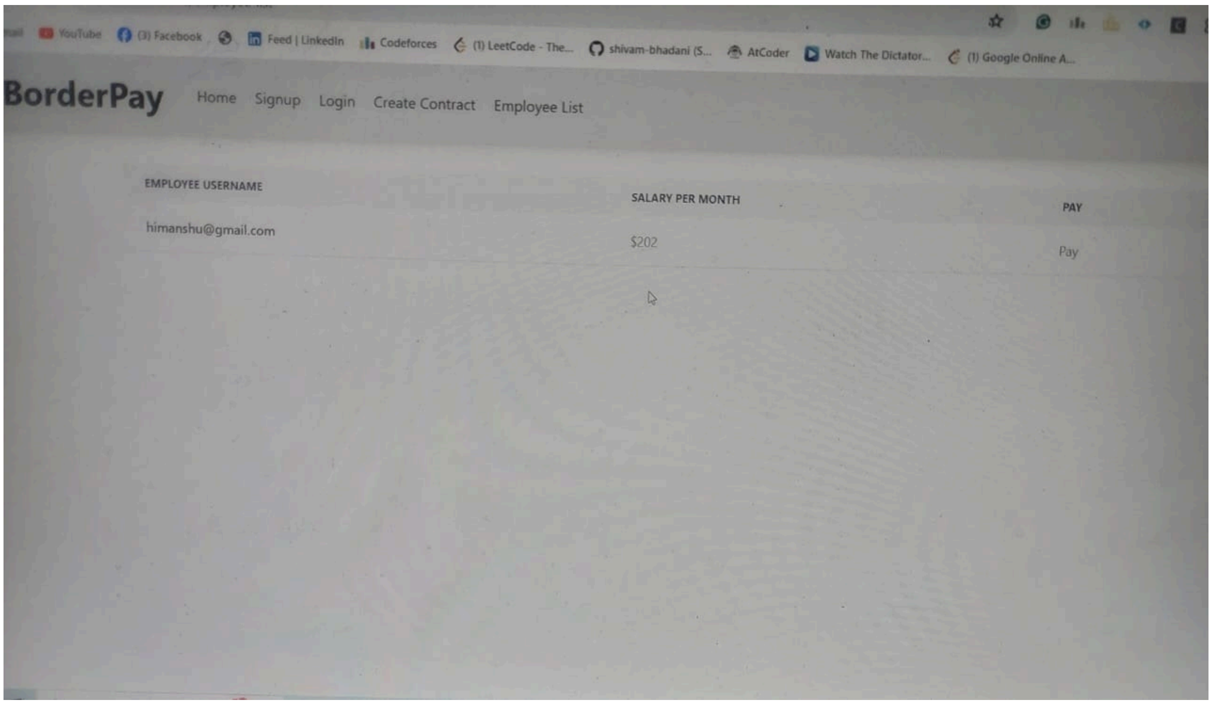
  Sign up page:



Login page:

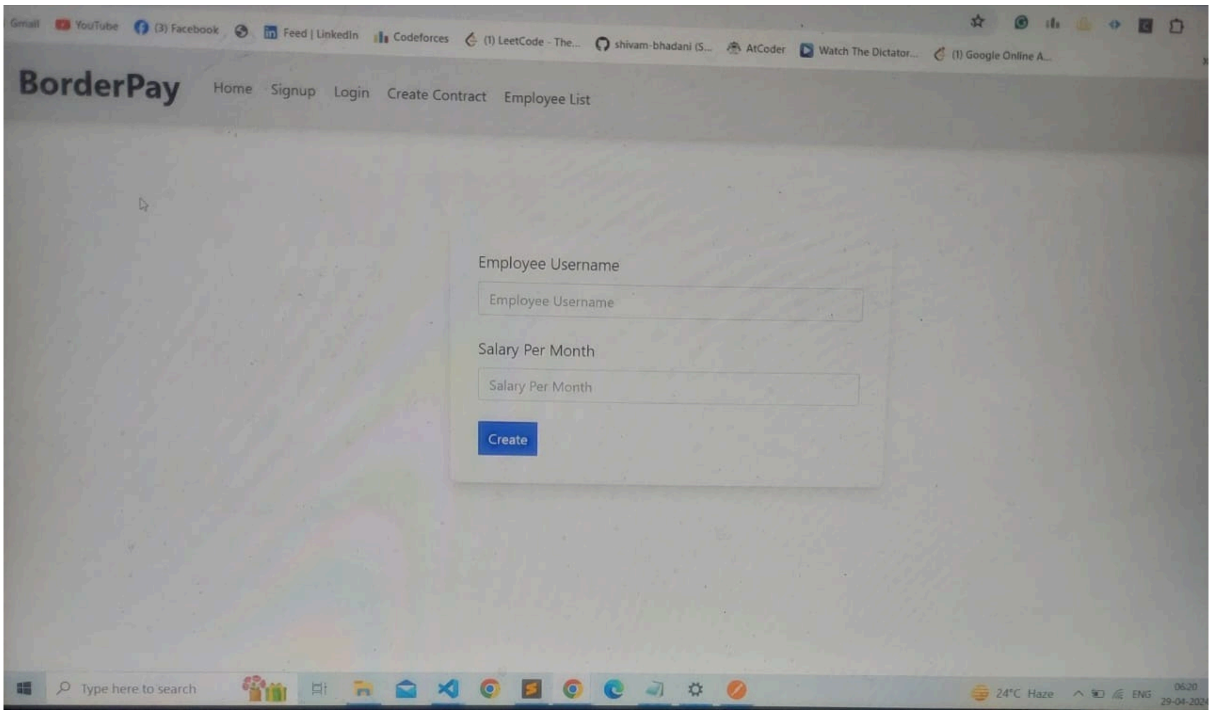


- **Dashboard -** For user signed in as employer. Employers can pay using pay button



EMPLOYEE USERNAME	SALARY PER MONTH	PAY
himanshu@gmail.com	$202	Pay

- **Contract page**: For user signed in as an employee. He can see the total money he has earned and can sign in contract. If the contract is already signed, it will display the contract



Backend:

Technologies used:

- Node.js
- Express.js
- Child Process

Core Functionalities of our Backend:

- **Chaincode Interaction:**The backend integrates with Hyperledger Fabric's chaincode through shell commands using the exec function from the child_process module. It dynamically constructs and executes commands to interact with the blockchain for various operations.

- **API Endpoints:**

    1. Initialization and Setup: The /GetAllContract endpoint includes commands to set up and reset the Hyperledger network, create a channel, deploy chaincode, and initialize the ledger.

    2. Contract and User Management: Endpoints like /GetAllContract, /UserExist, /ContractExist, /UserValid, /GetAllUser, /ReadContract, and /ReadUser provide RESTful access to corresponding chaincode functions. These allow for querying and managing contracts and users stored on the blockchain.

3. Transactional Functions: The /TransferMoney endpoint facilitates financial transactions between users (e.g., employer and employee), modifying their stored balances within the blockchain ledger.

Hyperledger:

- We have used 2 organizations where each organization has 1 peer
- One organization is used to represent employees and the other to represent employers/Banks
- We are joining the 2 organizations through 1 channel
- Chaincode is written in Go language

Some important functions in smart contract are -

1. **CreateUser**: Registers a new user in the system, storing necessary details like ID, password, account type, and balance.
2. **CreateContract**: Establishes a new employment contract between an employer and an employee with details such as salary and contract status.
3. **ReadUser**: Retrieves detailed information about a user from the ledger using the user ID.
4. **ReadContract**: Fetches detailed information about a contract from the ledger using the contract ID.
5. **TransferMoney**: Handles the financial transaction process, transferring salary payments from an employer's account to an employee's account.
6. **UserValid**: Validates a user’s credentials to ensure security during login or sensitive operations.
7. **UserExists and ContractExists**: Check if a user or contract already exists in the ledger, useful for validation before creation operations.
8. **GetAllContracts and GetAllUsers**: Retrieve lists of all contracts and users stored in the ledger, useful for administrative overview or audits.

Steps to set up hyperledger fabric:

- First clone the repo then go to test-network directory of fabric-samples folder.

- Bring up the network using the command: ./network.sh up

- Create a channel (named channel1) to join both the organizations:
  ./network.sh createChannel -c channel1

- Starting chaincode on channel:
  ./network.sh deployCC -ccn basic -ccp ../asset-transfer-basic/chaincode-go -c channel1 -ccl go

- Interacting with environment:
  export PATH=${PWD}/../bin:$PATH
  export FABRIC_CFG_PATH=$PWD/../config/

- Env variables for org1:
  export CORE_PEER_TLS_ENABLED=true
  export CORE_PEER_LOCALMSPID="Org1MSP"
  export CORE_PEER_TLS_ROOTCERT_FILE=${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt
  export CORE_PEER_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp
  export CORE_PEER_ADDRESS=localhost:7051

- Now we can run peer commands. We have tried to incorporate it in our Backend.