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:

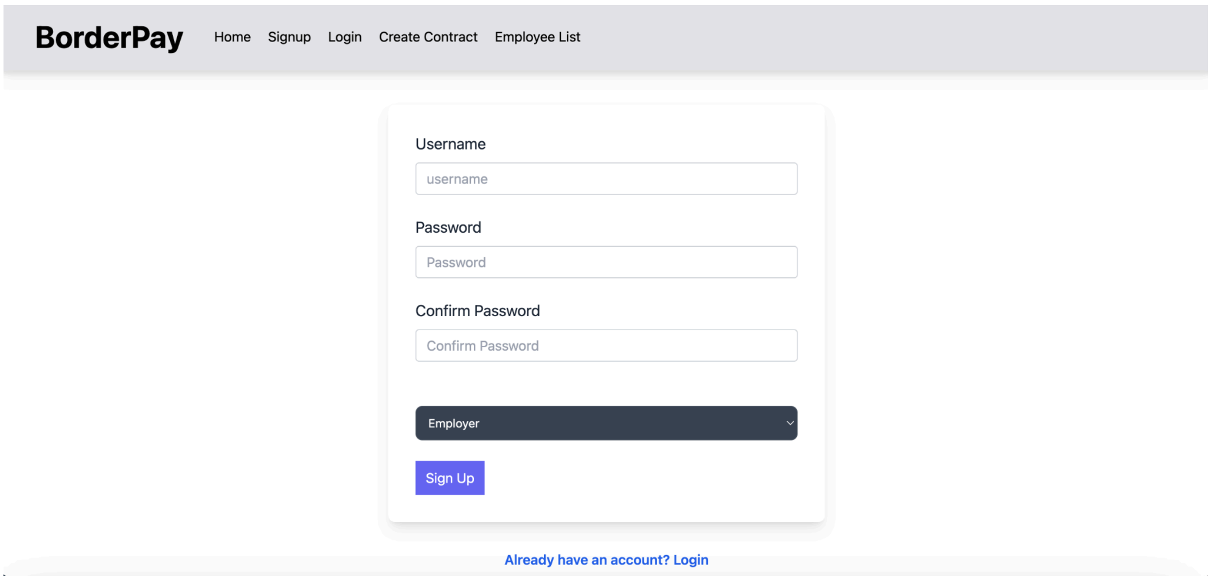
Technologies used:

- HTML
- Tailwind CSS
- JavaScript
- React
- Yup

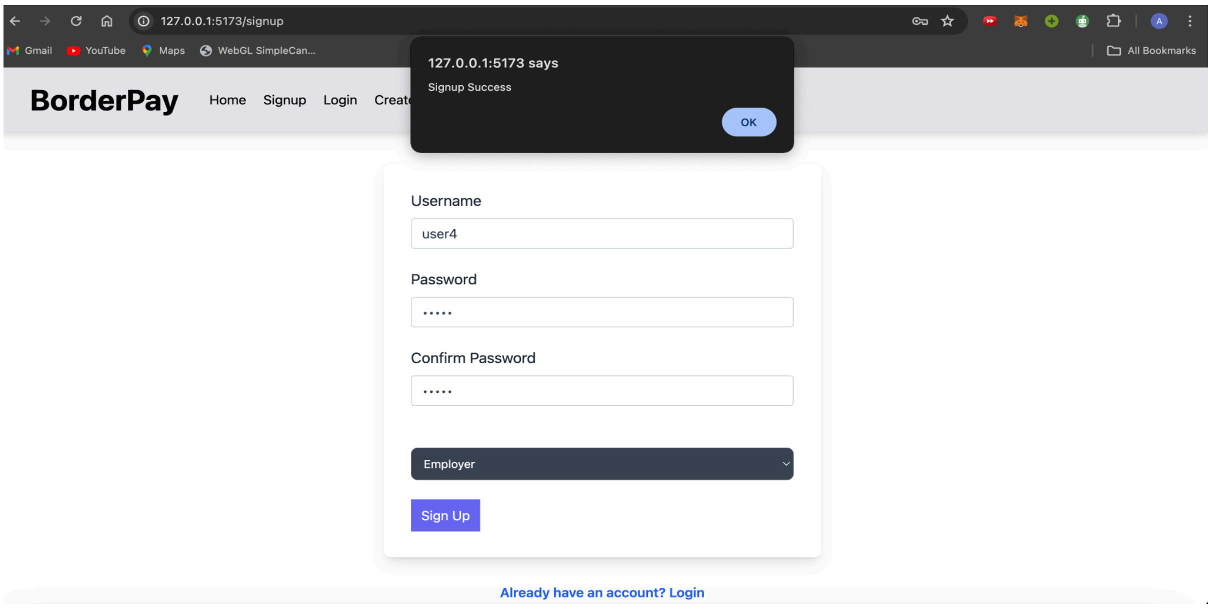
Major Components of Frontend are:

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:



Successful Signup:



127.0.0.1:5173/signup

127.0.0.1:5173 says

Signup Success

Username

user4

Password

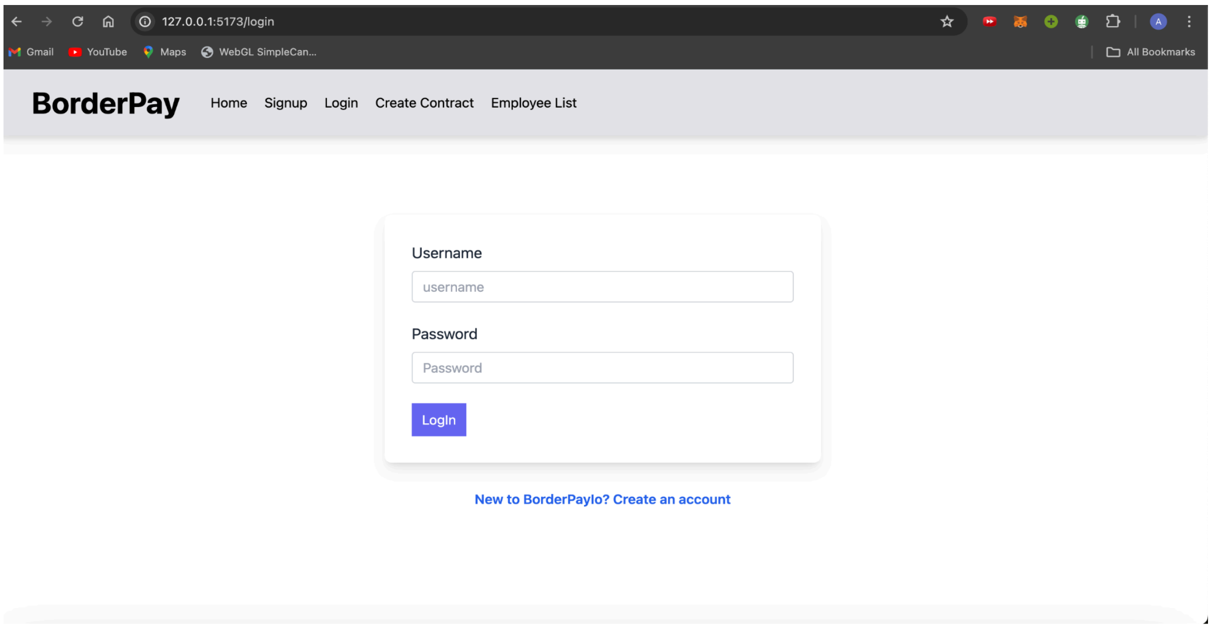
•••••

Confirm Password

•••••

Employer

Login page:



127.0.0.1:5173/login

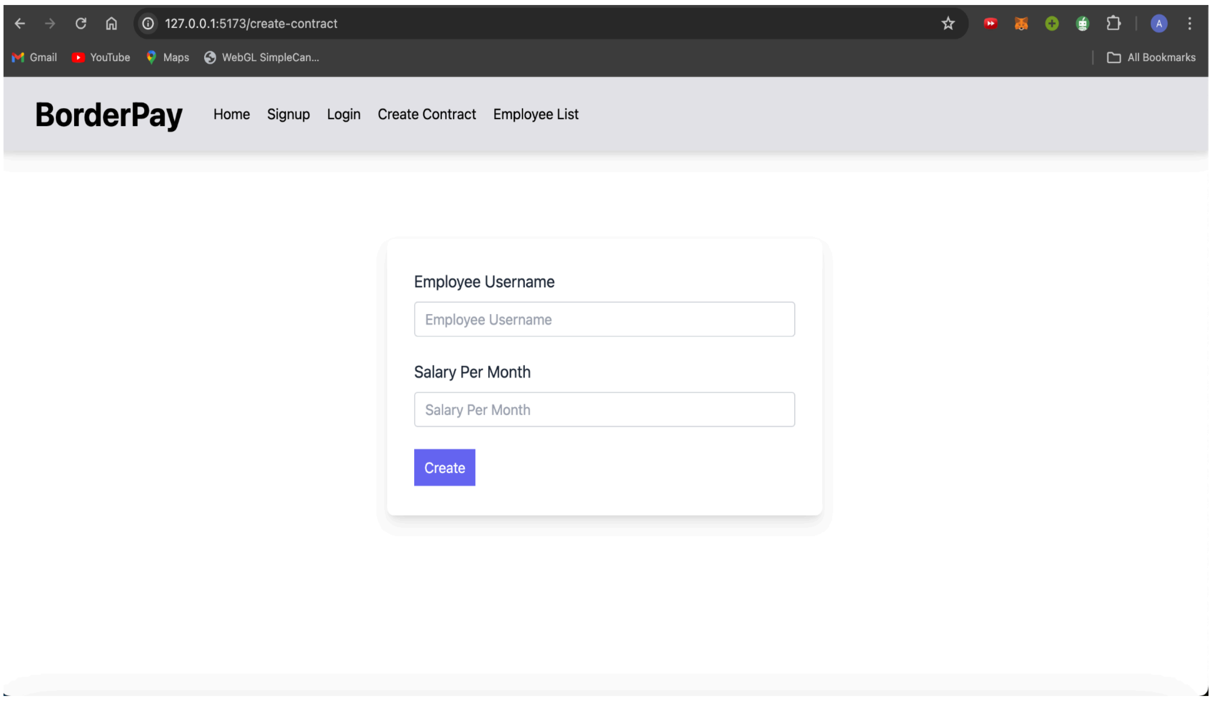
Username

username

Password

Password

Create Contract - For users signed in as employers. They can create new contracts by specifying employee username and their salary.



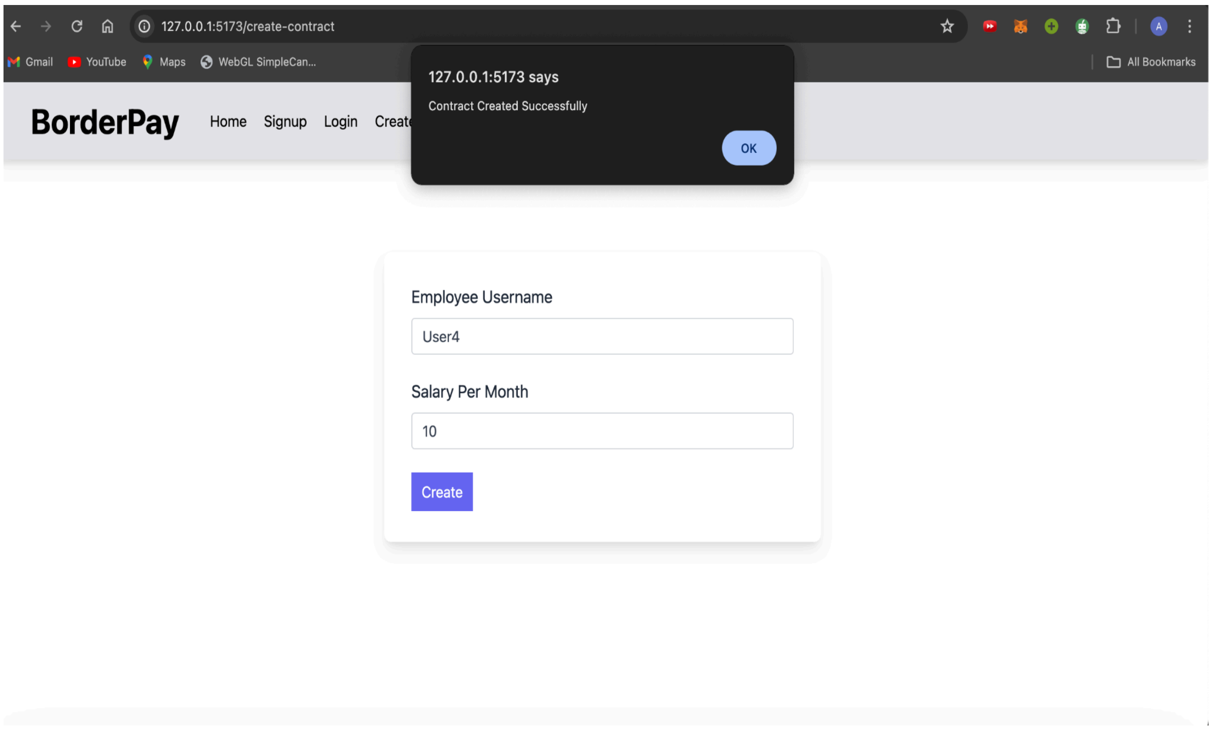
Employee Username

Employee Username

Salary Per Month

Salary Per Month

Successful contract creation:



127.0.0.1:5173 says

Contract Created Successfully

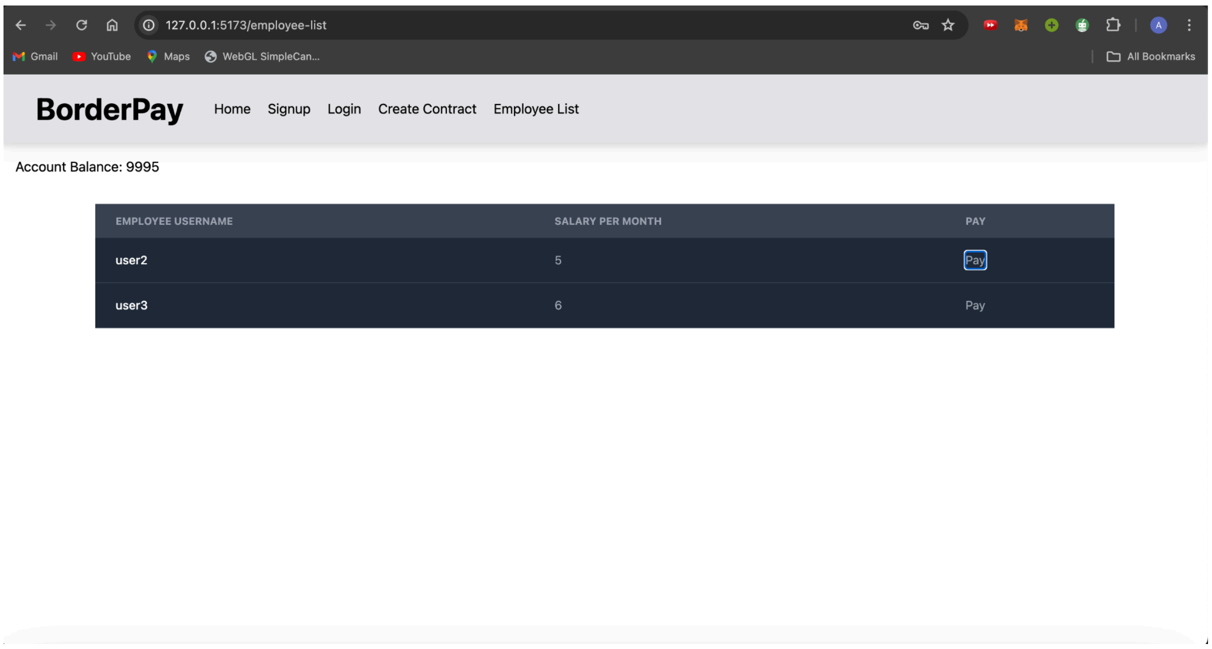
Employee Username

User4

Salary Per Month

10

Employee List:



127.0.0.1:5173/employee-list

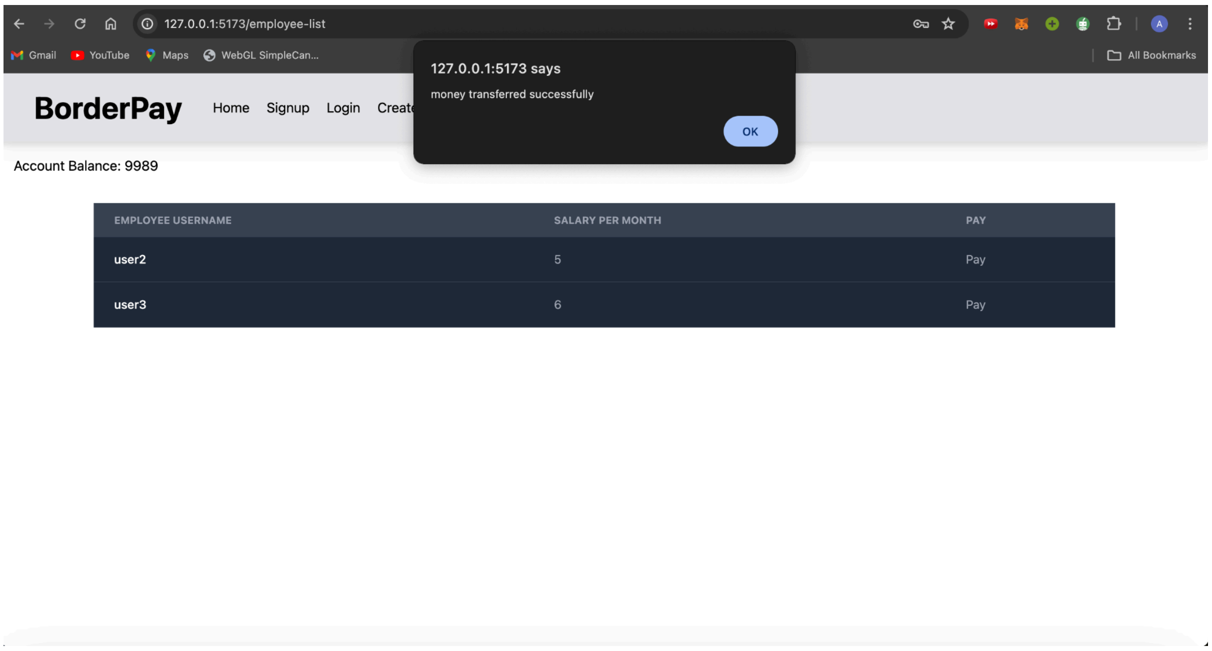
BorderPay Home Signup Login Create Contract Employee List

Account Balance: 9995

EMPLOYEE USERNAME	SALARY PER MONTH	PAY
user2	5	Pay
user3	6	Pay

1. For employers, they can see all the employees with whom they have a contract going on. There is a pay button through which they can transfer money from their bank to the desired employee. After successful transfer, money is deducted from the employer's account and transferred to employee's account
2. For employees, they can check their bank balance

Successful transfer: Money deducted from employer's account



127.0.0.1:5173/employee-list

127.0.0.1:5173 says

money transferred successfully

OK

BorderPay Home Signup Login Create

Account Balance: 9989

EMPLOYEE USERNAME	SALARY PER MONTH	PAY
user2	5	Pay
user3	6	Pay

Backend:

Technologies used:

- Node.js
- Express.js
- Hyperledger Fabric Gateway
- gRPC and @grpc/grpc-js (for communication protocol)

Core Functionalities of our Backend:

- **Ledger Initialization:**Initializes the ledger with a predefined set of data or assets.

- **User Management:**
  1. Create User: Handles the creation of new users, including storing their credentials and other pertinent details in the ledger.
  2. Read User: Retrieves details about specific users, useful for validation and verification processes.
  3. Validate User: Checks if user credentials are valid, which is critical for authentication and authorization tasks.

- **Contract Management:**
  1. Create Contract: Facilitates the creation of contracts between parties, such as employment contracts, with details about terms including salaries and operational status.
  2. Read Contract: Allows for querying details of specific contracts, which is essential for oversight and management.

- **Transactional Operations:** Manages monetary transactions between entities, such as payroll processing or contractual payments, reflecting updates in the ledger.

- **Bank Information Handling:** Retrieves banking or financial details associated with a specific ID, supporting financial operations and verifications using readBank

- **Reporting and Analytics:**
  1. Get All Users: Provides a comprehensive list of all users registered in the system, which can be used for audits, reporting, or administrative purposes.
  2. Get All Contracts: Offers a complete overview of all contracts recorded in the ledger, essential for compliance, monitoring, and management.

- **API Endpoints:**
  The backend application we have set up using Hyperledger Fabric and Node.js with Express includes several API endpoints to interact with the blockchain. Here's a detailed list of the API endpoints and their purposes:

1. **POST /initLedger**: Initializes the ledger with a predefined set of assets or data as per the smart contract's InitLedger function.

2. **POST /createUser**: Creates a new user in the ledger with details such as ID, password, and account type.

3. **POST /createContrac**t: Creates a new employment contract between an employer and an employee, including details like salary and contract status.

4. **POST /readUser**: Retrieves a user’s details based on the provided user ID.

5. **POST /readBank**: Retrieves bank details associated with a given ID, likely related to a user or a corporate entity.

6. **POST /readContract**: Retrieves details of a specific contract using the contract ID.

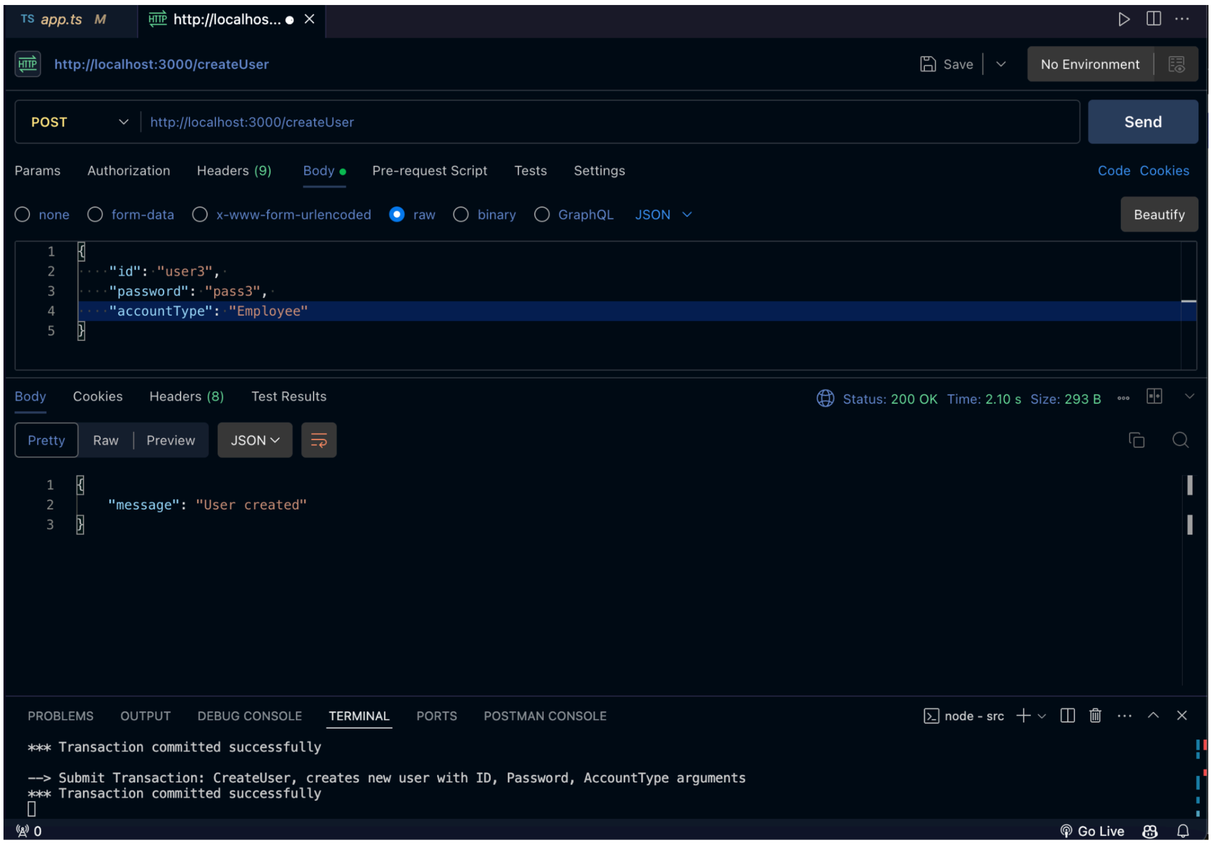
7. **POST /transferMoney**: Transfers money from one entity to another, likely within the context of fulfilling payment terms in a contract.

8. **POST /UserValid**: Validates a user’s credentials to ensure that they are a registered and active user in the system.

9. **GET /getAllContracts**: Retrieves all contracts currently stored in the ledger.

10. **GET /getAllUsers**: Retrieves all user profiles stored in the ledger.

- **An example of endpoint testing of API implemented in Hyperledger Fabric through Postman:**



POST http://localhost:3000/createUser

```
{
    "id": "user3",
    "password": "pass3",
    "accountType": "Employee"
}
```

Status: 200 OK Time: 2.10 s Size: 293 B

```
{
    "message": "User created"
}
```

```
*** Transaction committed successfully

--> Submit Transaction: CreateUser, creates new user with ID, Password, AccountType arguments
*** Transaction committed successfully
```

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.

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

- Create a channel (mychannel in our case) to join both the organizations:
  ```
  ./network.sh createChannel -c mychannel
  ```

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

- After successful deployment of chaincode, the hyperledger will be able to connect with the backend.