Getting Started with this App

This README explain the function in this application.

Prerequisites

Make sure you have the following installed on your machine:

- Node.js (which includes npm)
- Visual Studio Code
- MetaMask extension on browser

Steps to Open the App

Direct access to netlify

https://jacky-020-cv-app.netlify.app/

Deploy locally (may fails because of environmental issue)

1. **Download the Project**

   - Download the project folder and extract it to your desired location.

2. **Open the Project in VS Code**

   - Launch Visual Studio Code.
   - Open the downloaded project folder.

3. **Navigate to the Project Directory**

   - In the terminal, change directory to the folder containing the /src folder
   - It should be the folder where you cloned the repository using Git clone

4. **Install Dependencies**

   - Run the following command to install all necessary dependencies:

     ```
     npm install
     ```

5. **Start the Application**

   - After the installation is complete, start the application with:

     ```
     npm start
     ```

   - This will launch the app in your default web browser.

6. **View the App**
    - Open your browser and go to http://localhost:3000 to view the application.

Troubleshooting

- If you encounter any issues, ensure that all dependencies are installed correctly and that your Node.js version is compatible with the project.
- `node -v` to check nodejs version, the version I uses is `20.15.1`

General View

Motivation

It is common that employee fakes or exaggerates work experience in their CV. It is not easy for employers to validate among hundreds of CV. This is CV validation app provides additional validation features on top of job-seeking applications like LinkedIn. It utilizes the past immutability and decentralized properties of blockchain such that every work experience record in the app, upon comfirmation, will not be artificially changed. A peer comment function help ensures employers are acting honestly in approving employee's work experience record

General Features:

- As an *employee*, you can send work experience proposals to an *employer* and wait for approval.
- As an *employer*, you can approve received proposals.
- Upon approving a proposal, the *employer* gains reputation points as a sign of being more trustworthy.
- As an *employer*, you can comment on other *employers* when necessary (peer comments).
- For all users, you can view *employees*' work experience records. The decentralized and transparent properties of blockchain ensure confidentiality.
- For all users, you can view peer comments about *employers*. This ensures that *employers* are responsible when approving work experience records; otherwise, they may be condemned by other *employers*.

This app must be used with the MetaMask plugin. When using the app, please switch to the corresponding MetaMask account. All actions will be sent using this MetaMask account, and there are no other login functions available.

Pages:

- **Registration (Starting Page)**: For registering a MetaMask account as one of the two types: *employee* or *employer*.
- **Show Records**: Displays submitted/received work experience records for the current account.
- **Propose Work Experience**: Allows *employee* accounts to send proposed work experience records to *employers*.
- **Search Employers**: Displays a list of employer accounts. Users can click on a particular employer to see their peer comments. If the user is an *employer*, they can proceed to comment if they have enough reputation points.
- **Search Employees**: For checking the work experience records of other *employees*.

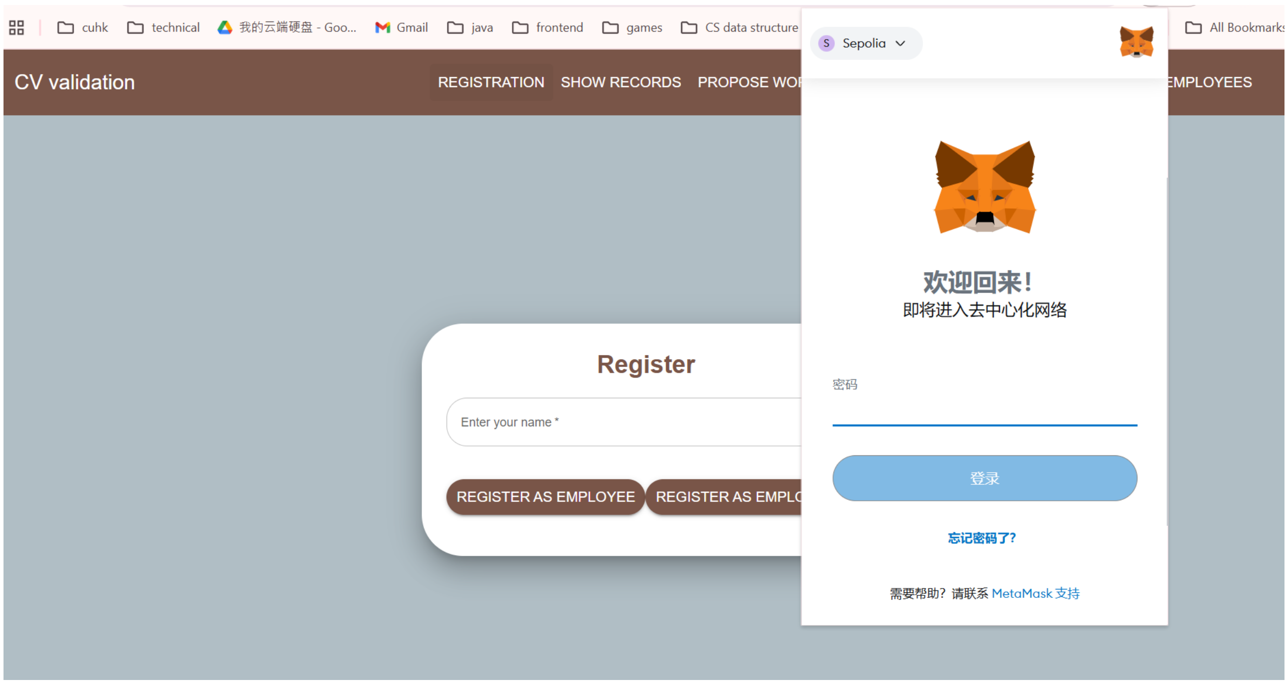
Detail workflow

Notes

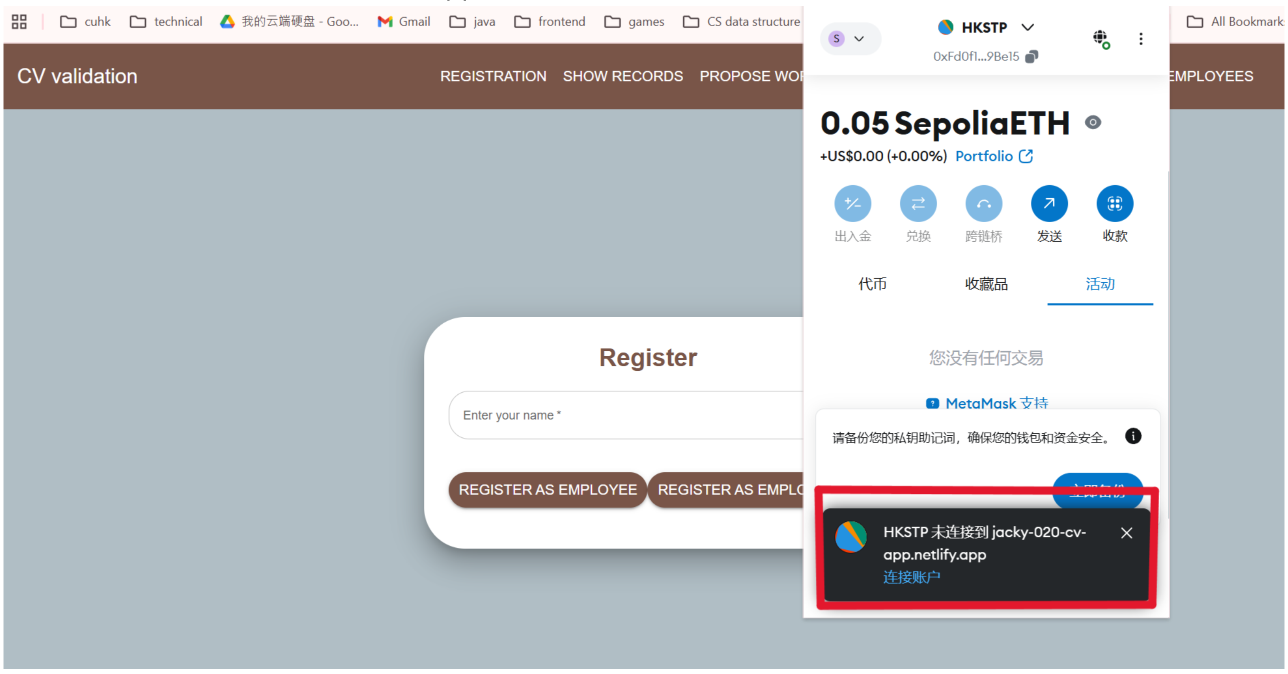
- You need to refresh the page after switching accounts. There is no function to constantly detects account status because of efficiency issue. This is intentionally as in real scenerio most user only need one account for this app.
- If you visit the netlify version, the refresh and address bar is not working properly, please use the top navbar.
- Some function are available for *employee* or *employer* only, pay attention to the workflow below.
- Make sure you choose the right Metamask accounts.

preparation

1. login Metamask account



2. connect Metamask account to the app



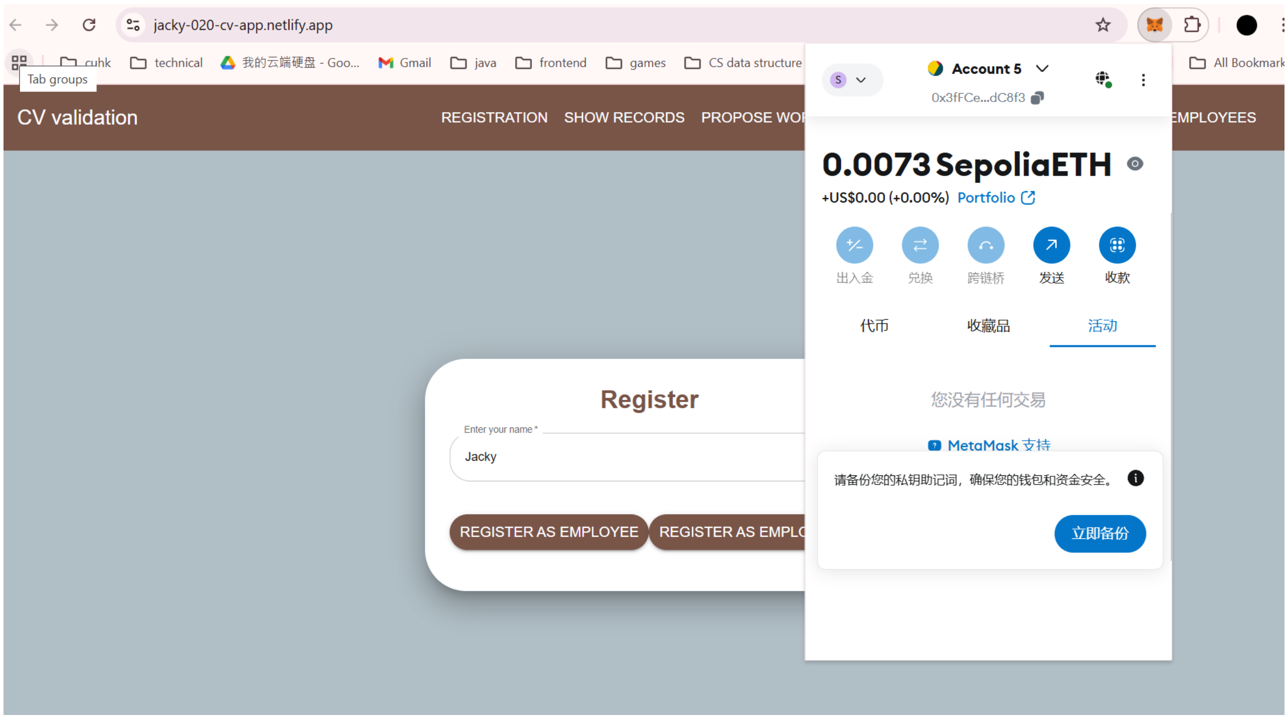
- The app only works when you have switched to a connected Metamask account

registration

1. click on the **REGISTRATION** on the top navbar to switch to registation page.
2. Enter the account name.
3. To register your account as *employee*, click on the left tab (**REGISTER AS EMPLOYEE**) . To register your account as *employer*, click on the right tab (**REGISTER AS EMPLOYER**).
4. Comfirm transaction with your Metamask account, makes sure you have enough ether.
5. The button will turn grey during the transaction, please wait patiently.
6. Upon successful registration, the button returns normal and an alert `${name} has been registered as ${role} successfully!` pops out.
7. This account can then be used in other function

- Each Metamask account can only register once, either as *employee* or *employer*. This is to encourage user to be responsible to their account.

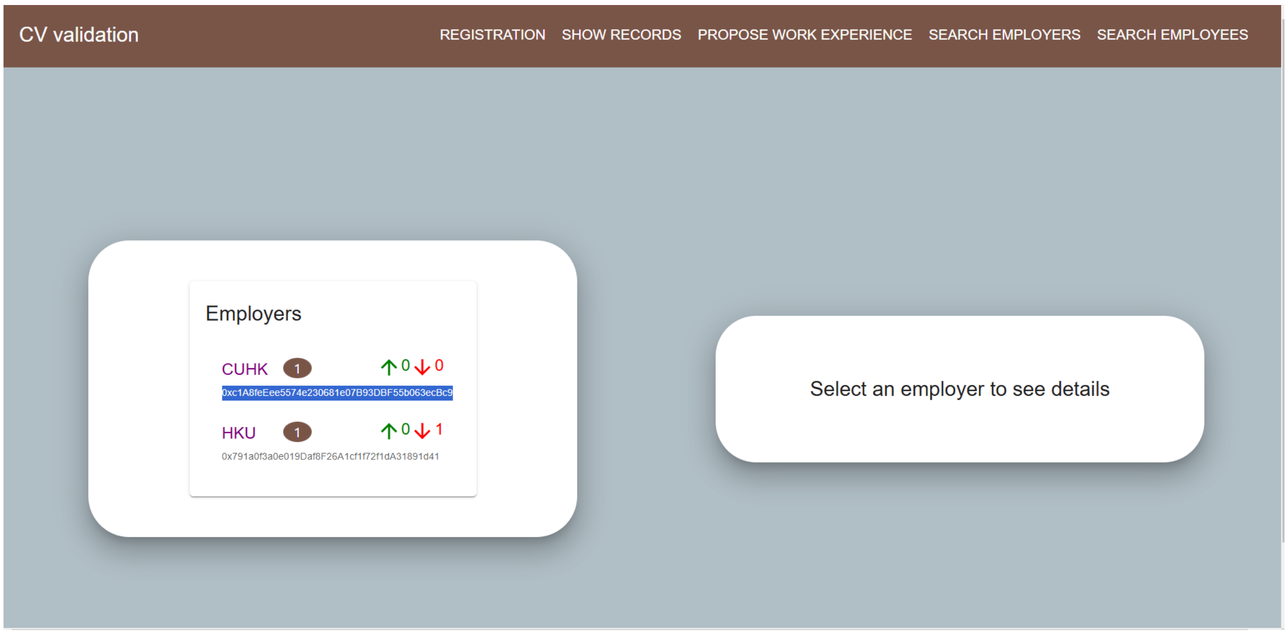
- For testing, recommends to at least prepare 2 accounts, one as an *employee* and another as *employer*



Propose

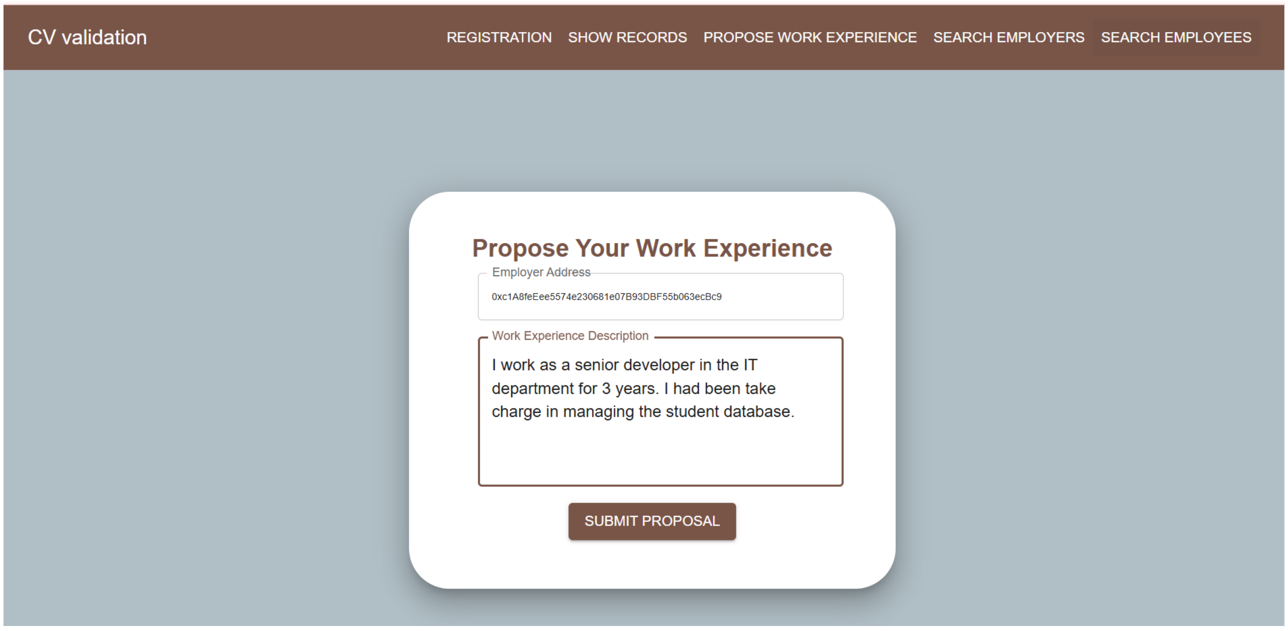
switch to an employee account

1. Click **SEARCH EMPLOYERS** on the navbar to the employer page.
2. Choose an employer that you want to propose a work experience record for validation, copy the account address (the text with smaller size).


CV validation
REGISTRATION
SHOW RECORDS
PROPOSE WORK EXPERIENCE
SEARCH EMPLOYERS
SEARCH EMPLOYEES
Employers
CUHK
1
0
0
0xc1A8feEee5574e230681e07B93DBF55b063ecBc9
HKU
1
0
1
0x791a0f3a0e019Daf8F26A1cf1f72f1dA31891d41
Select an employer to see details

3. Click **PROPOSE** on the navbar to the propose page.

4. Paste the copied account address, enter the propose work description and click **SUBMIT PROPOSAL**



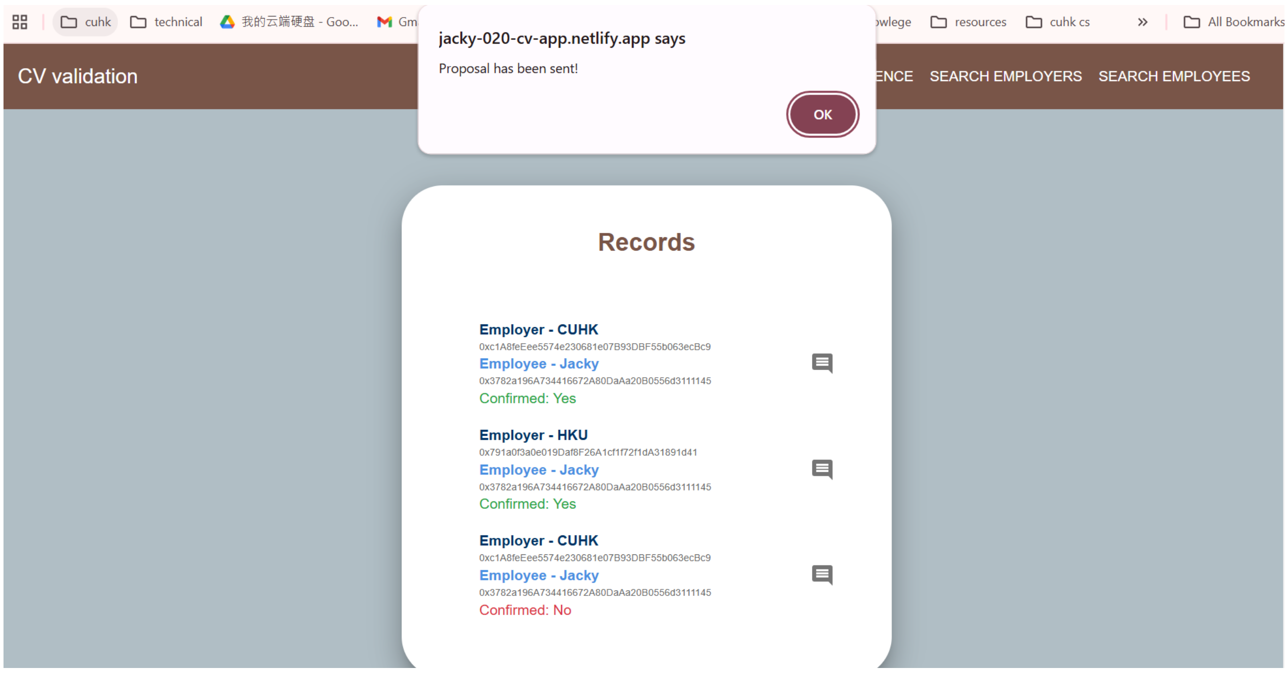
Propose Your Work Experience

Employer Address
0xc1A8feEee5574e230681e07B93DBF55b063ecBc9

Work Experience Description
I work as a senior developer in the IT department for 3 years. I had been take charge in managing the student database.

5. The propose work experience can be viewed in **SHOW RECORDS** page by the sender (*employee*) and receiver (*employer*) as **Comfirmed: No**.

- It takes time for the proposal to settle in the blockchain, you need to wait for a few seconds before it appears in the **SHOW RECORDS** page. An alert will show up when it finished.
- This waiting time is insignificant in real scenerio as you should not expect employer to immediate approve your proposal.



jacky-020-cv-app.netlify.app says

Proposal has been sent!

Records

Employer - CUHK
0xc1A8feEee5574e230681e07B93DBF55b063ecBc9
Employee - Jacky
0x3782a196A734416672A80DaAa20B0556d3111145
Confirmed: Yes

Employer - HKU
0x791a0f3a0e019Daf8F26A1cf1f72f1dA31891d41
Employee - Jacky
0x3782a196A734416672A80DaAa20B0556d3111145
Confirmed: Yes

Employer - CUHK
0xc1A8feEee5574e230681e07B93DBF55b063ecBc9
Employee - Jacky
0x3782a196A734416672A80DaAa20B0556d3111145
Confirmed: No

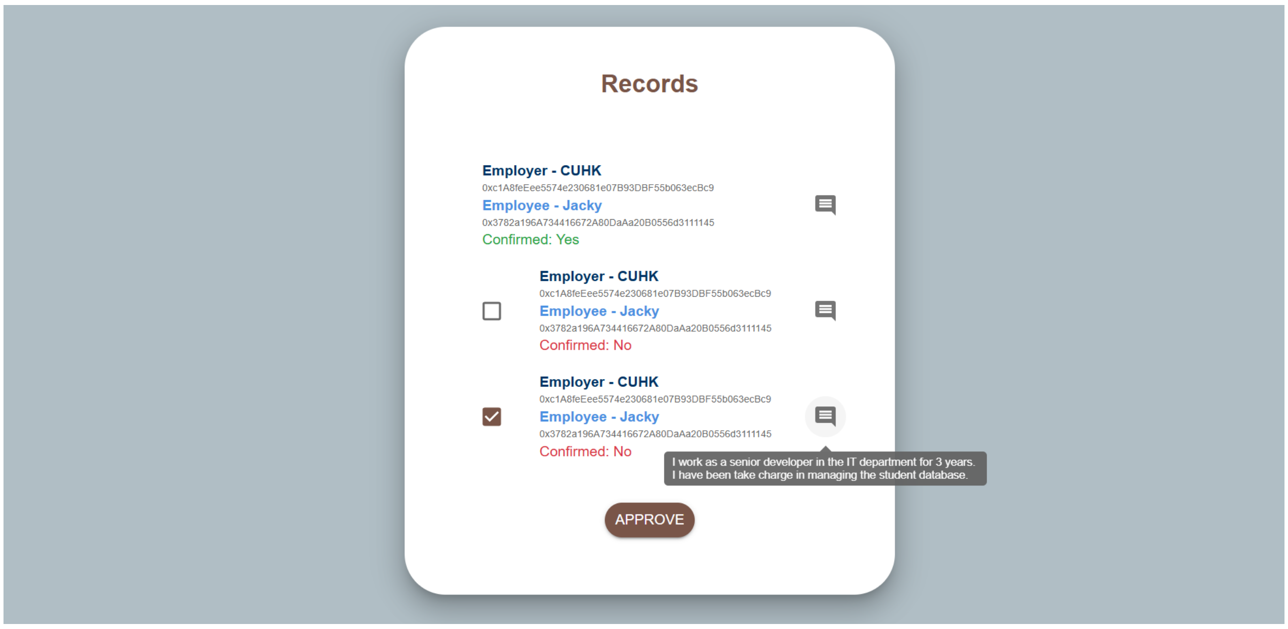
Approve Records

switch to an employer account

1. Notes the difference in UI for *employee* and *employer*: *employer* has check box and approve button.
2. Hover over the comment icon to see the work description (available for both *employer* and *employee* account).
3. Click on those records that you want to approve, then click **APPROVE**.

4. Wait for the transaction settles, revisit the **SHOW RECORDS** page, you will see the the record approved will be indicated as **Comfirmed: Yes**.

- approve record from sender's account (*employee*)



Records

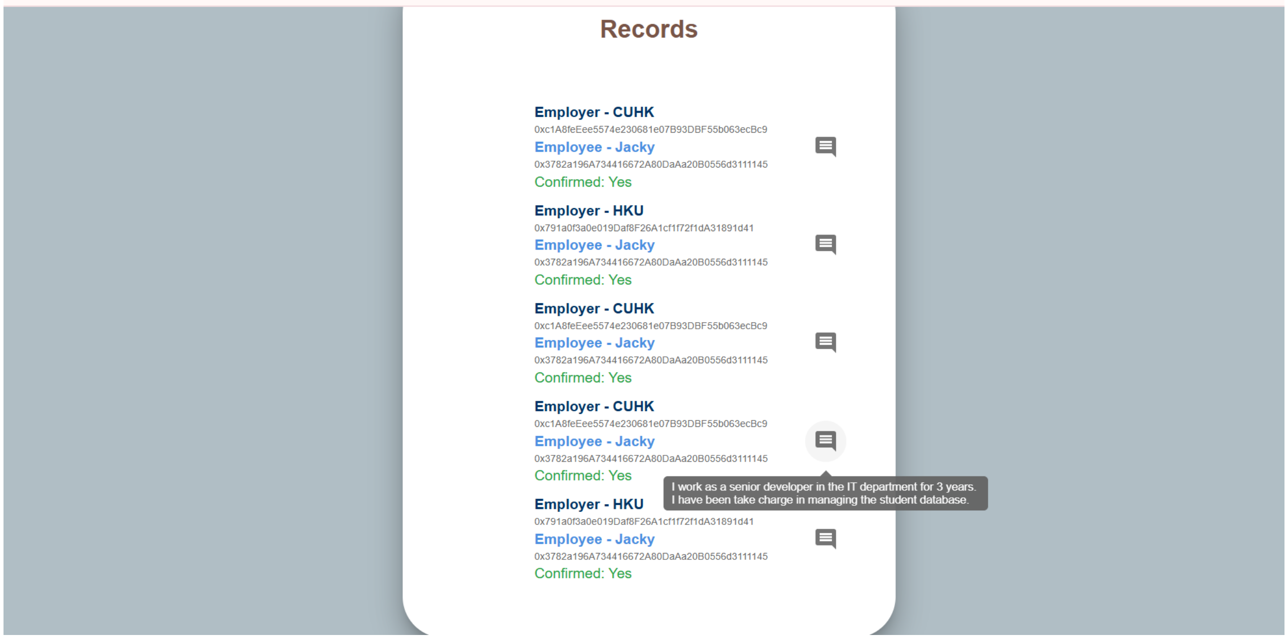
Employer - CUHK
0xc1A8feEee5574e230681e07B93DBF55b063ecBc9
Employee - Jacky
0x3782a196A734416672A80DaAa20B0556d3111145
Confirmed: Yes

Employer - CUHK
0xc1A8feEee5574e230681e07B93DBF55b063ecBc9
Employee - Jacky
0x3782a196A734416672A80DaAa20B0556d3111145
Confirmed: No

Employer - CUHK
0xc1A8feEee5574e230681e07B93DBF55b063ecBc9
Employee - Jacky
0x3782a196A734416672A80DaAa20B0556d3111145
Confirmed: No

I work as a senior developer in the IT department for 3 years.
I have been take charge in managing the student database.

- record after approval from sender's account (*employee*)



Records

Employer - CUHK
0xc1A8feEee5574e230681e07B93DBF55b063ecBc9
Employee - Jacky
0x3782a196A734416672A80DaAa20B0556d3111145
Confirmed: Yes

Employer - HKU
0x791a0f3a0e019Daf8F26A1cf1f72f1dA31891d41
Employee - Jacky
0x3782a196A734416672A80DaAa20B0556d3111145
Confirmed: Yes

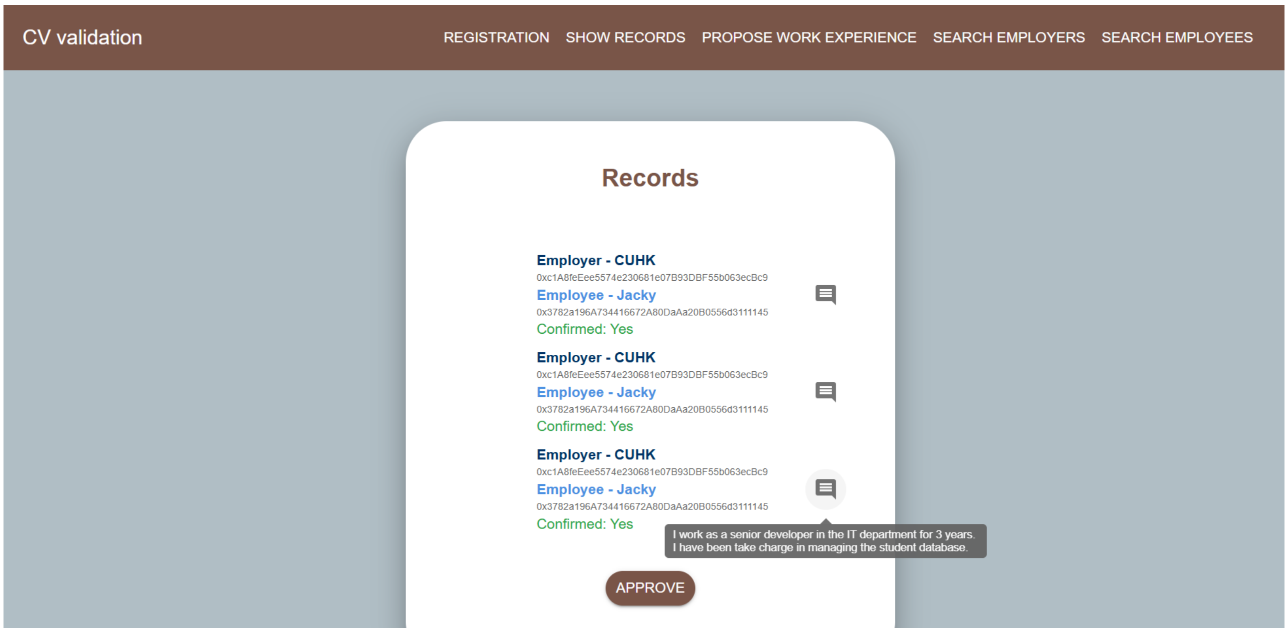
Employer - CUHK
0xc1A8feEee5574e230681e07B93DBF55b063ecBc9
Employee - Jacky
0x3782a196A734416672A80DaAa20B0556d3111145
Confirmed: Yes

Employer - CUHK
0xc1A8feEee5574e230681e07B93DBF55b063ecBc9
Employee - Jacky
0x3782a196A734416672A80DaAa20B0556d3111145
Confirmed: Yes

I work as a senior developer in the IT department for 3 years.
I have been take charge in managing the student database.

Employer - HKU
0x791a0f3a0e019Daf8F26A1cf1f72f1dA31891d41
Employee - Jacky
0x3782a196A734416672A80DaAa20B0556d3111145
Confirmed: Yes

- record after approval from receiver's account (*employer*)



Records

Employer - CUHK
0xc1A8feEee5574e230681e07B93DBF55b063ecBc9
Employee - Jacky
0x3782a196A734416672A80DaAa20B0556d3111145
Confirmed: Yes

Employer - CUHK
0xc1A8feEee5574e230681e07B93DBF55b063ecBc9
Employee - Jacky
0x3782a196A734416672A80DaAa20B0556d3111145
Confirmed: Yes

Employer - CUHK
0xc1A8feEee5574e230681e07B93DBF55b063ecBc9
Employee - Jacky
0x3782a196A734416672A80DaAa20B0556d3111145
Confirmed: Yes

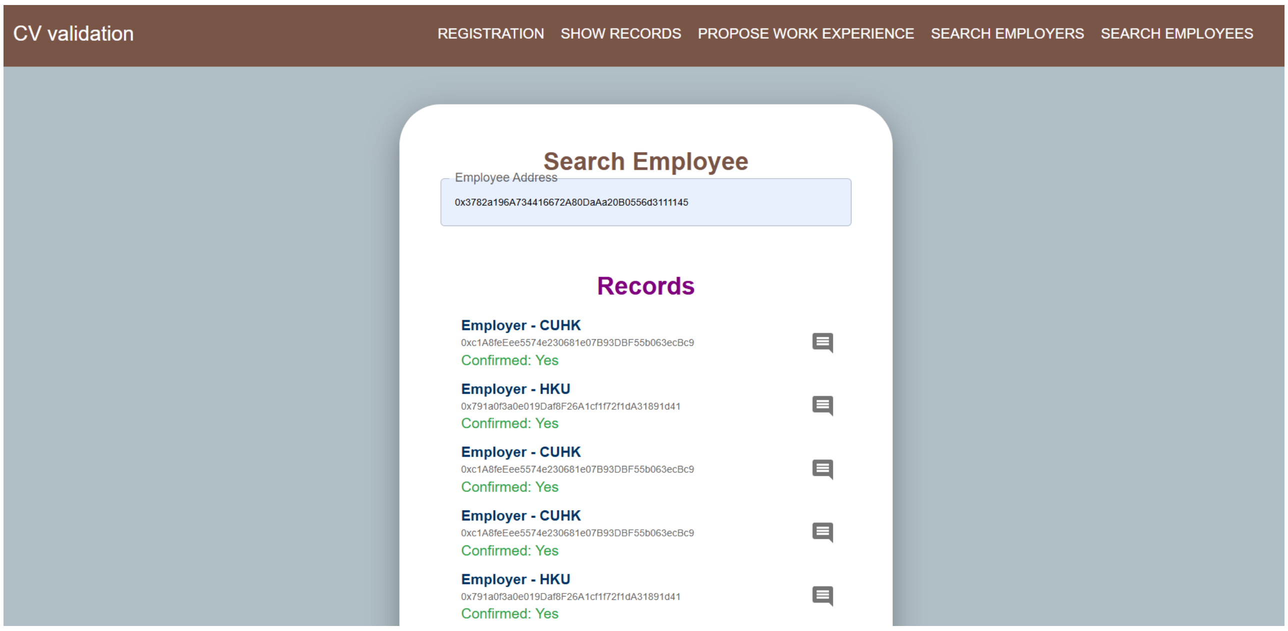
I work as a senior developer in the IT department for 3 years.
I have been take charge in managing the student database.

Views any employee record

may switch to any account

1. Copy the account address of the *employee* you want to view. For convenient, you may copy the address of your *employee* account.
2. Mouse over from the text field, you should see the record list if the address is correct
3. Notes that only the comfirmed records are shown, the *employee* name is also hidden for privacy

- Notes that there is no way to get the list of *employee*, you are supposed to get the *employee* name and address through other routes, e.g. in CV or social media like LinkedIn. This ensures that all visitors have get permission from that *employee* to protect privacy.



Search Employee

Employee Address
0x3782a196A734416672A80DaAa20B0556d3111145

Records

Employer - CUHK
0xc1A8feEee5574e230681e07B93DBF55b063ecBc9
Confirmed: Yes

Employer - HKU
0x791a0f3a0e019Daf8F26A1cf1f72f1dA31891d41
Confirmed: Yes

Employer - CUHK
0xc1A8feEee5574e230681e07B93DBF55b063ecBc9
Confirmed: Yes

Employer - CUHK
0xc1A8feEee5574e230681e07B93DBF55b063ecBc9
Confirmed: Yes

Employer - HKU
0x791a0f3a0e019Daf8F26A1cf1f72f1dA31891d41
Confirmed: Yes

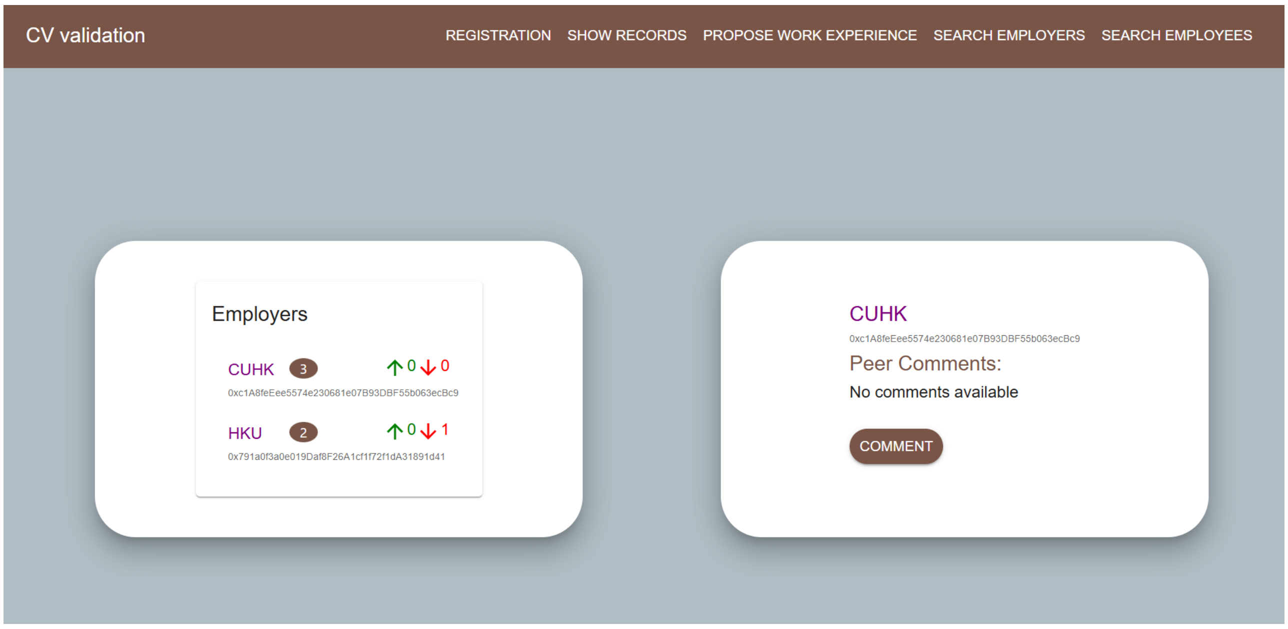
Employer list

switch to employer account

1. Observe the employer list.

2. The number in the **brown ellipse** is the **reputation points**. It equals to the number of records approved by that *employer*, representing the trustworthiness of that *employer*. It is also the upper limit of peer comment made by that *employee*.
3. The number to the right of the **up arrow** and **down arrow** are the number of **up vote** and **down vote** made by other employers respectively.
4. Click on the **COMMENT** button to make peer comment (in an *employer* account), make sure you have more reputation points than the peer comment you made. If not, approve some.
5. Enter the comment and choose either **upvote** or **downvote**, then click **SEND COMMENT**.
6. Wait and you will see the peer comment appears.

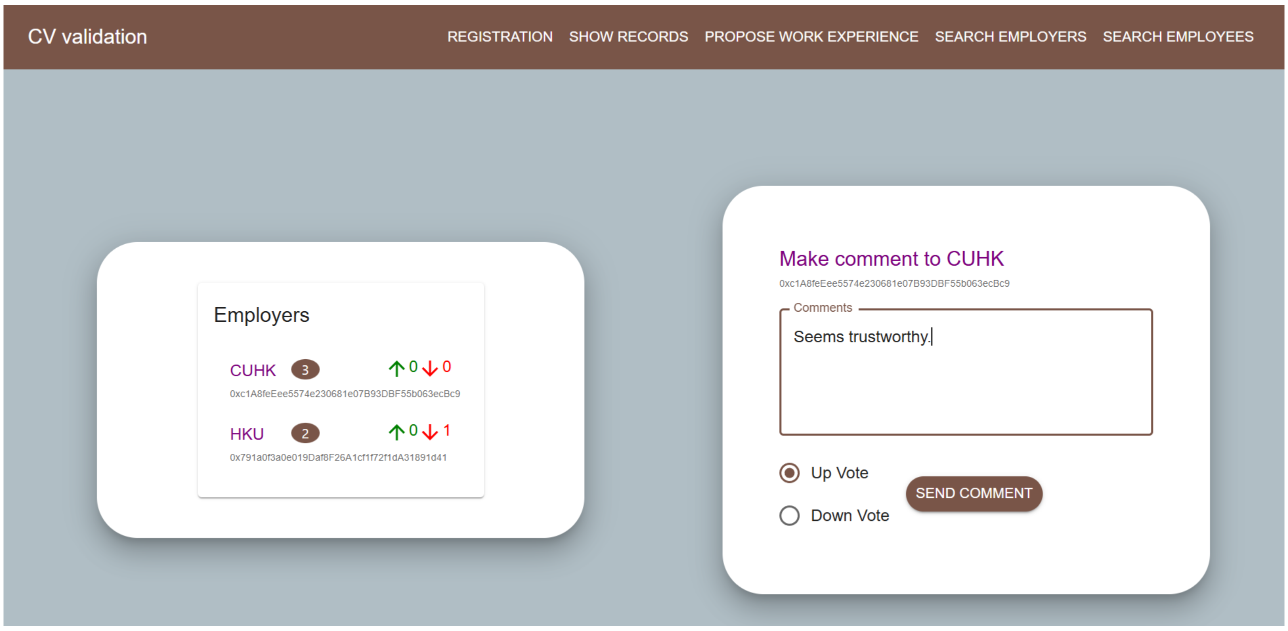
- Initial view from an *employer* account



CUHK	3	↑ 0	↓ 0
0xc1A8feEee5574e230681e07B93DBF55b063ecBc9			
HKU	2	↑ 0	↓ 1
0x791a0f3a0e019Daf8F26A1cf1f72f1dA31891d41			

CUHK
0xc1A8feEee5574e230681e07B93DBF55b063ecBc9
Peer Comments:
No comments available

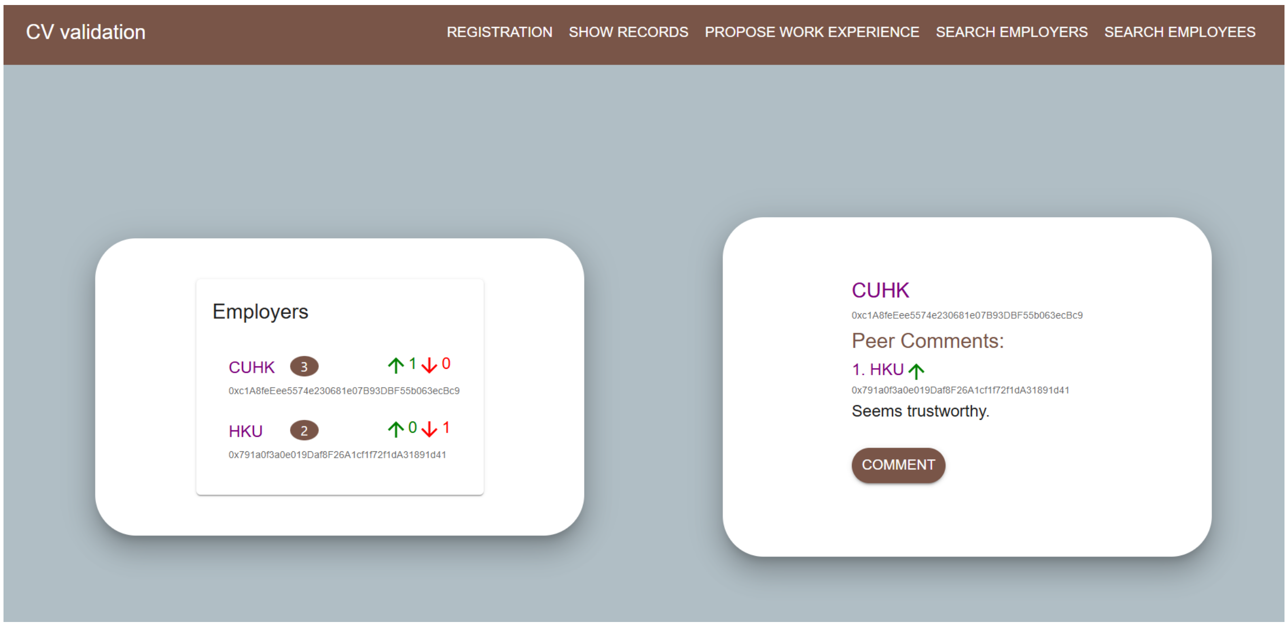
- Send peer comment with an *employer* account



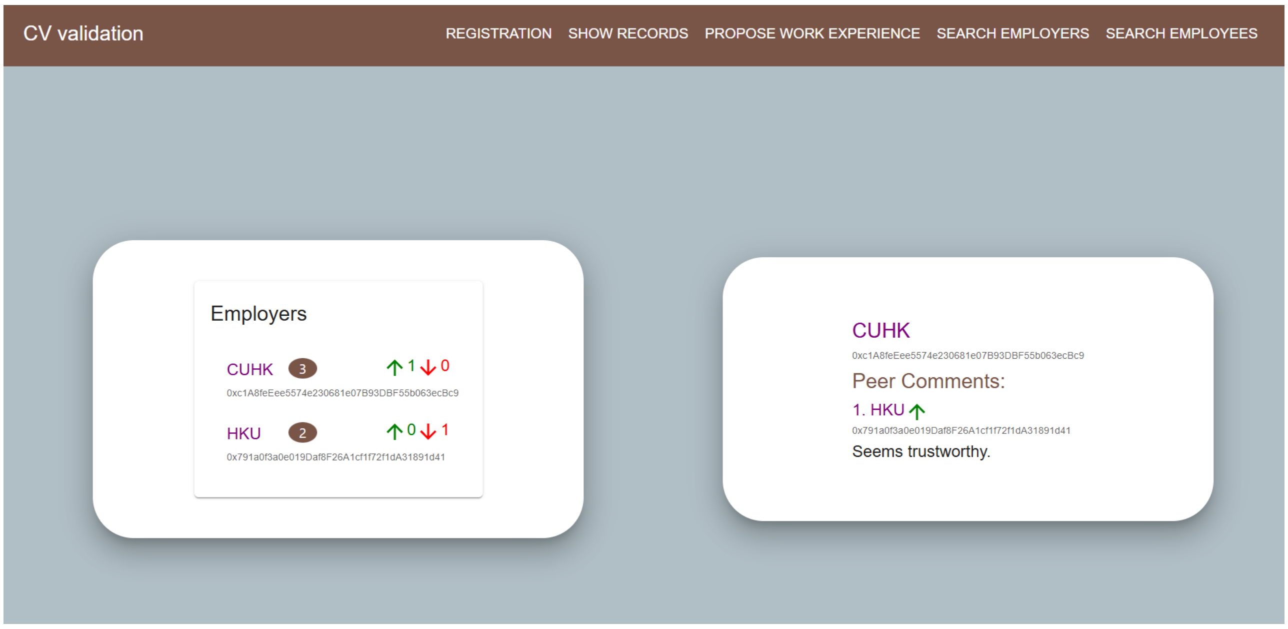
CUHK	3	↑ 0	↓ 0
0xc1A8feEee5574e230681e07B93DBF55b063ecBc9			
HKU	2	↑ 0	↓ 1
0x791a0f3a0e019Daf8F26A1cf1f72f1dA31891d41			

Make comment to CUHK
0xc1A8feEee5574e230681e07B93DBF55b063ecBc9
Comments: Seems trustworthy.
[✓ Up Vote] Down Vote

- view from an *employer* account (click on one of the *employers*)



- view from an *employee* account (click on one of the *employers*)


CV validation
REGISTRATION
SHOW RECORDS
PROPOSE WORK EXPERIENCE
SEARCH EMPLOYERS
SEARCH EMPLOYEES
Employers
CUHK 3 ↑1 ↓0
0xc1A8feEee5574e230681e07B93DBF55b063ecBc9
HKU 2 ↑0 ↓1
0x791a0f3a0e019Daf8F26A1cf1f72f1dA31891d41
CUHK
0xc1A8feEee5574e230681e07B93DBF55b063ecBc9
Peer Comments:
1. HKU ↑
0x791a0f3a0e019Daf8F26A1cf1f72f1dA31891d41
Seems trustworthy.

Mechanism

Solidity code

```
// reference Only
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.7.0 <0.9.0;

contract WorkExperienceValidation {

    struct Employee {
        address addr;
        string name;
        uint256[] submittedExperienceIds; // ref to universalExperienceRecord
    }
    struct Employer {
        address addr;
        string name;
        uint256[] receivedExperienceIds; // ref to universalExperienceRecord
        uint256 reputationPoints; // Points for confirming work experiences
        uint256[] peerCommentIds; // ref to peerCommentRecord
        uint256 commentMadeCounter; // Number of comments made to other employers
        uint256 upVote; // Count of upvotes received
        uint256 downVote; // Count of downvotes received
    }

    struct WorkExperienceEntry {
        string description; // work description
        address employer; // receiver's address
        string employerName;
        address employee; // sender's address
        string employeeName;
        uint256 entryId; // ref to universalExperienceRecord
        bool confirmed;
    }

    struct PeerComment {
        bool upVote;
        string comment;
        string commentorName;
        address commentorAddress;
    }

    mapping(address => Employee) internal employees; // stores all employees
    mapping(address => Employer) internal employers; // stores all employers
    mapping(uint256 => WorkExperienceEntry) internal universalExperienceRecord; //
stores all work experience records
    mapping(uint256 => PeerComment) internal peerCommentRecord; // stores all
peerComment
    address[] internal employerAddresses; // store employer address
    uint256 internal nextEntryId; // Track the next work experience record entry
ID
    uint256 internal nextPeerCommentId; // Track next PeerComment ID

    event EmployeeRegistered(address indexed employeeAddr, string name);
    event EmployerRegistered(address indexed employerAddr, string name);
    event WorkExperienceAdded(uint256 indexed entryId, address indexed employee,
address indexed employer);
    event WorkExperienceConfirmed(uint256 indexed entryId, address indexed
employer);
    event PeerCommentAdded(address indexed to, address indexed from, string
comment, bool upVote);

    constructor() {
        nextPeerCommentId = 1; // Initialize to 1 to avoid zero ID usage
    }

    // Function to register an employee
    function registerEmployee(string memory name) external {
        require(employees[msg.sender].addr == address(0), "You have already
registered as an employee");
        require(bytes(name).length > 0, "Name cannot be empty");

        employees[msg.sender] = Employee({
            addr: msg.sender,
            name: name,
            submittedExperienceIds: new uint256[](0)
        });
        emit EmployeeRegistered(msg.sender, name);
    }

    // Function to register an employer
    function registerEmployer(string memory name) external {
        require(employers[msg.sender].addr == address(0), "You have already
registered as an employer");
        require(bytes(name).length > 0, "Name cannot be empty");

        employers[msg.sender] = Employer({
            addr: msg.sender,
            name: name,
            receivedExperienceIds: new uint256[](0),
            reputationPoints: 0,
            peerCommentIds: new uint256[](0),
            commentMadeCounter: 0,
            upVote: 0,
            downVote: 0
        });
        employerAddresses.push(msg.sender);
        emit EmployerRegistered(msg.sender, name);
    }

    // Function to propose a work experience
    function proposeExperience(address to, string calldata workExperience)
external {
        require(employees[msg.sender].addr != address(0), "Only employees can
submit work experience records");
        require(employers[to].addr != address(0), "You are not sending the message
to a valid employer");

        WorkExperienceEntry memory newEntry = WorkExperienceEntry({
            description: workExperience,
            employer: to,
            employerName: employers[to].name,
            employee: msg.sender,
            employeeName: employees[msg.sender].name,
            entryId: nextEntryId,
            confirmed: false
        });

        universalExperienceRecord[nextEntryId] = newEntry;

        employees[msg.sender].submittedExperienceIds.push(nextEntryId);
        employers[to].receivedExperienceIds.push(nextEntryId);

        emit WorkExperienceAdded(nextEntryId, msg.sender, to);
        nextEntryId++;
    }

    // Function for the employer to confirm a work experience entry
    function confirmWorkExperience(uint256 entryId) external {
        WorkExperienceEntry storage entry = universalExperienceRecord[entryId];
        require(entry.employer != address(0), "This is not a valid entry");
        require(entry.employer == msg.sender, "Only the employer can confirm this
entry");
        require(!entry.confirmed, "Entry already confirmed");

        entry.confirmed = true;
        employers[msg.sender].reputationPoints += 1; // Increase reputation points
for confirmation

        emit WorkExperienceConfirmed(entryId, msg.sender);
    }

    // Function to get employee records
    function getEmployeeRecords(address employeeAddr) external view returns
(WorkExperienceEntry[] memory) {
        require(employees[employeeAddr].addr != address(0), "This is not a valid
employee account");

        uint256[] memory experienceIds =
employees[employeeAddr].submittedExperienceIds;
        WorkExperienceEntry[] memory records = new WorkExperienceEntry[]
(experienceIds.length);

        for (uint256 i = 0; i < experienceIds.length; i++) {
            records[i] = universalExperienceRecord[experienceIds[i]];
        }

        return records;
    }

    // Function to get employer records for their employees' experiences
    function getEmployerRecords(address employerAddr) external view returns
(WorkExperienceEntry[] memory) {
        require(employers[employerAddr].addr != address(0), "This is not a valid
employer account");

        uint256[] memory recordIds =
employers[employerAddr].receivedExperienceIds;
        WorkExperienceEntry[] memory records = new WorkExperienceEntry[]
(recordIds.length);

        for (uint256 i = 0; i < recordIds.length; i++) {
            records[i] = universalExperienceRecord[recordIds[i]];
        }
        return records;
    }

    // Function to get a paginated list of employers, page start from 1
    function getEmployerList(uint256 page) external view returns (Employer[]
memory) {
        uint256 pageSize = 10; // Define how many employers per page
        uint256 startIndex = (page - 1) * pageSize;
        uint256 endIndex = startIndex + pageSize > employerAddresses.length ?
employerAddresses.length : startIndex + pageSize;

        require(startIndex < employerAddresses.length, "Page index out of
bounds");

        Employer[] memory employerList = new Employer[](endIndex - startIndex);

        for (uint256 i = startIndex; i < endIndex; i++) {
            employerList[i - startIndex] = employers[employerAddresses[i]];
        }

        return employerList;
    }

    // Function that allows an employer to comment on another employer
    function peerComment(string calldata commentText, bool upVote, address to)
external {
        require(employers[msg.sender].addr != address(0), "This is not a valid
employer account");
        require(employers[msg.sender].commentMadeCounter <
employers[msg.sender].reputationPoints,
                "You have made more comments than your reputation points");
        require(to != msg.sender, "You cannot comment on yourself"); // Prevent
self-commenting

        PeerComment memory newComment = PeerComment({
            upVote: upVote,
            comment: commentText,
            commentorName: employers[msg.sender].name,
            commentorAddress: msg.sender
        });

        peerCommentRecord[nextPeerCommentId] = newComment;
        employers[to].peerCommentIds.push(nextPeerCommentId);
        nextPeerCommentId++; // Increment the peerCommentId
        employers[msg.sender].commentMadeCounter += 1; // Increment the comment
counter

        if (upVote) {
            employers[to].upVote += 1;
        } else {
            employers[to].downVote += 1;
        }

        emit PeerCommentAdded(to, msg.sender, commentText, upVote);
    }

    // Function to get comments made to an employer
    function getPeerComments(address employerAddr) external view returns
(PeerComment[] memory) {
        uint256[] memory commentIds = employers[employerAddr].peerCommentIds;
        PeerComment[] memory comments = new PeerComment[](commentIds.length);

        for (uint256 i = 0; i < commentIds.length; i++) {
            comments[i] = peerCommentRecord[commentIds[i]];
        }

        return comments;
    }

    // Function that return the account type
    function checkRole() external view returns (string memory) {
        if(employers[msg.sender].addr != address(0)){
            return "employer";
        }
        if(employees[msg.sender].addr != address(0)){
            return "employee";
        }
        return "none";
    }
}
```

Registration

Frontend

- /src/compomnents/Registration.js

```
const handleRegistration = async (role) => {
      // Validate name input
      if (!name) {
         setError('Name cannot be empty.');
         return;
      }

      const contract = await connectEthereum();

      try {
         setLoading(true);
         // Call the appropriate contract method based on role
         const tx = (role === 'employee') ? await contract.registerEmployee(name)
: await contract.registerEmployer(name);
         const result = await tx.wait();

         // Check transaction status
         if (result.status === 1) {
               alert(`${name} has been registered as ${role} successfully!`);
               setName(''); // Clear input field
               setError(''); // Clear any previous errors
         } else {
               setError('Transaction rejected.'); // Handle transaction rejection
         }
      } catch (e) {
         console.error(e);
         setError(`Error registering as ${role}. Please try again.`); // Handle
any errors
      } finally {
         setLoading(false); // Reset loading state
      }
   };
```

- When click on the registration button, `handleRegistration` is called, which then call either the `registerEmployee` or `registerEmployer` in solidity corresponding to the button clicked.

Backend

```

    struct Employee {
        address addr;
        string name;
        uint256[] submittedExperienceIds; // ref to universalExperienceRecord
    }

    struct Employer {
        address addr;
        string name;
        uint256[] receivedExperienceIds; // ref to universalExperienceRecord
        uint256 reputationPoints; // Points for confirming work experiences
        uint256[] peerCommentIds; // ref to peerCommentRecord
        uint256 commentMadeCounter; // Number of comments made to other employers
        uint256 upVote; // Count of upvotes received
        uint256 downVote; // Count of downvotes received
    }

    mapping(address => Employee) internal employees; // stores all employees
    mapping(address => Employer) internal employers; // stores all employers
    mapping(uint256 => WorkExperienceEntry) internal universalExperienceRecord; //
stores all work experience records
    mapping(uint256 => PeerComment) internal peerCommentRecord; // stores all
peerComment
    address[] internal employerAddresses; // store employer address
    uint256 internal nextEntryId; // Track the next work experience record entry
ID
    uint256 internal nextPeerCommentId; // Track next PeerComment ID

// Function to register an employee
    function registerEmployee(string memory name) external {
        require(employees[msg.sender].addr == address(0), "You have already
registered as an employee");
        require(bytes(name).length > 0, "Name cannot be empty");

        employees[msg.sender] = Employee({
            addr: msg.sender,
            name: name,
            submittedExperienceIds: new uint256[](0)
        });
        emit EmployeeRegistered(msg.sender, name);
    }

    // Function to register an employer
    function registerEmployer(string memory name) external {
        require(employers[msg.sender].addr == address(0), "You have already
registered as an employer");
        require(bytes(name).length > 0, "Name cannot be empty");

        employers[msg.sender] = Employer({
            addr: msg.sender,
            name: name,
            receivedExperienceIds: new uint256[](0),
            reputationPoints: 0,
            peerCommentIds: new uint256[](0),
            commentMadeCounter: 0,
            upVote: 0,
            downVote: 0
        });
        employerAddresses.push(msg.sender);
        emit EmployerRegistered(msg.sender, name);
    }
```

- `Employee`:
  - `uint256[] submittedExperienceIds`: corresponds to the work experience records submitted
- `Employer`:
  - `uint256 [] receivedExperienceIds`: corresponds to the work experience records received
  - `uint256[] peerCommentIds`: corresponds to peer comment received
- The above keep an array of id, which are the references to the actual storage of experience record entries and peer comment in `mapping universalExperienceRecord` and `mapping peerCommentRecord`
- Both `registerEmployee` and `registerEmployer` new an `Employee` or `Employer` with the `name` and `addr` set and all other fields empty.
- `registerEmployer` also stores the address in `address[] employerAddresses` for the retrieval of employer list.

Full frontend code

```
/**
 * Registration Component
 *
 * This component provides a user interface for registering users as either
 * an employee or an employer within the application. It includes an input
 * field for entering the user's name. There are 2 buttons for registering as
 * employee and employer respectively.
 *
 */

const Registration = () => {
    // State variables for name input, loading status, and error messages
    const [name, setName] = useState('');
    const [loading, setLoading] = useState(false);
    const [error, setError] = useState('');

    // Handles registration for employee or employer based on the button clicked
    const handleRegistration = async (role) => {
        // Validate name input
        if (!name) {
            setError('Name cannot be empty.');
            return;
        }

        const contract = await connectEthereum();

        try {
            setLoading(true);
            // Call the appropriate contract method based on role
            const tx = (role === 'employee') ? await
contract.registerEmployee(name) : await contract.registerEmployer(name);
            const result = await tx.wait();

            // Check transaction status
            if (result.status === 1) {
                alert(`${name} has been registered as ${role} successfully!`);
                setName(''); // Clear input field
                setError(''); // Clear any previous errors
            } else {
                setError('Transaction rejected.'); // Handle transaction rejection
            }
        } catch (e) {
            console.error(e);
            setError(`Error registering as ${role}. Please try again.`); // Handle
any errors
        } finally {
            setLoading(false); // Reset loading state
        }
    };

    return (
        <StyledContainer maxWidth="sm">
            <StyledCard elevation={16}>
                <FormContainer>
                    <Typography component="h1" variant="h5" align="center"
color='primary' sx={{ fontWeight: 600 }}>
                        Register
                    </Typography>
                    <form>
                        {/* Text field for entering the account name */}
                        <StyledTextField
                            variant="outlined"
                            margin="normal"
                            required
                            id="name"
                            label="Enter your name"
                            name="name"
                            autoComplete="name"
                            autoFocus
                            value={name}
                            onChange={(e) => setName(e.target.value)}
                        />
                        {error && <p style={{ color: 'red' }} aria-
live="assertive">{error}</p>}
                        <div>
                            {/* Button for registering as an employee */}
                            <StyledButton
                                type="button"
                                variant="contained"
                                color="primary"
                                onClick={() => handleRegistration('employee')}
                                disabled={loading}
                            >
                                {loading ? 'Registering...' : 'Register as
Employee'}
                            </StyledButton>
                            {/* Button for registering as an employer */}
                            <StyledButton
                                type="button"
                                variant="contained"
                                color="primary"
                                onClick={() => handleRegistration('employer')}
                                disabled={loading}
                            >
                                {loading ? 'Registering...' : 'Register as
Employer'}
                            </StyledButton>
                        </div>
                    </form>
                </FormContainer>
            </StyledCard>
        </StyledContainer>
    );
}
```

Propose

Frontend

- /src/compomnents/Propose.js

```
    // State variables for employer address and work description
    const [employerAddress, setEmployerAddress] = useState('');
    const [workDescription, setWorkDescription] = useState('');

    // Handles form submission
    const handleSubmit = async (event) => {
        event.preventDefault(); // Prevent default form submission behavior

        // Connect to Ethereum and get the contract
        const contract = await connectEthereum();

        try {
            // Call the contract method to propose work experience
            const tx = await contract.proposeExperience(employerAddress,
workDescription);
            await tx.wait(); // Wait for the transaction to be mined
            alert('Proposal has been sent!'); // Notify the user of success
        } catch (error) {
            console.error(error); // Log the error for debugging
            alert('Error in sending proposal. Please try again. Make sure you are
logged in using your employee metamask account.'); // Notify the user of the error
        }
    };
```

- get the `employerAddress` and `workDescription` from text field
- When click on the submit button, `handleSubmit` is called, which then call `proposeExperience` in solidity corresponding to the button clicked.

Backend

```
    struct WorkExperienceEntry {
        string description; // work description
        address employer; // receiver's address
        string employerName;
        address employee; // sender's address
        string employeeName;
        uint256 entryId; // ref to universalExperienceRecord
        bool confirmed;
    }
   mapping(uint256 => WorkExperienceEntry) internal universalExperienceRecord; //
stores all work experience records

   uint256 internal nextEntryId; // Track the next work experience record entry ID

   // Function to propose a work experience
   function proposeExperience(address to, string calldata workExperience) external
{
      require(employees[msg.sender].addr != address(0), "Only employees can submit
work experience records");
      require(employers[to].addr != address(0), "You are not sending the message
to a valid employer");

      WorkExperienceEntry memory newEntry = WorkExperienceEntry({
         description: workExperience,
         employer: to,
         employerName: employers[to].name,
         employee: msg.sender,
         employeeName: employees[msg.sender].name,
         entryId: nextEntryId,
         confirmed: false
      });

      universalExperienceRecord[nextEntryId] = newEntry;

      employees[msg.sender].submittedExperienceIds.push(nextEntryId);
      employers[to].receivedExperienceIds.push(nextEntryId);

      emit WorkExperienceAdded(nextEntryId, msg.sender, to);
      nextEntryId++;
   }
```

- `WorkExperienceEntry`:
  - `description`: main body of the work description
  - `employer`, `employerName`, `employee`, `employeeName`: info of sender and receiver
  - `entryId`: key id in the `mapping universalExperienceRecord` where this entry is stored
  - `comfirmed`: signify if the record is comfirmed, it is set to be false in this `proposeExperience` function
- `proposeExperience`:
  - Check if the sender and receiver are valid
  - new a `WorkExperienceEntry` with `comfirmed` set to `false`, then stored in `mapping universalExperienceRecord`
  - store the entry id `nextEntryId` in `submittedExperienceIds` of the sender (*employee*) and `receivedExperienceIds` of the receiver (*employer*). This allows the sender and receiver to retrieve the entry.
  - increment the `nextEntryId` to prepare for the next entry.

Full frontend code

```
/**
 * Propose Work Experience Page
 *
 * This component renders a form for proposing work experience, allowing users to
 * enter an employer address and a work description. Upon submission, the proposal
 * is sent to the specified employer account as an "unconfirmed"
 * record. Users can later check this proposal on the "Show Records" page,
 * accessible by both the sender (employee) and the receiver (employer).
 *
 * Important: If the sender's account is not recognized as an employee, the
 * transaction will fail.
 */

const Propose = () => {
    // State variables for employer address and work description
    const [employerAddress, setEmployerAddress] = useState('');
    const [workDescription, setWorkDescription] = useState('');

    // Handles form submission
    const handleSubmit = async (event) => {
        event.preventDefault(); // Prevent default form submission behavior

        // Connect to Ethereum and get the contract
        const contract = await connectEthereum();

        try {
            // Call the contract method to propose work experience
            const tx = await contract.proposeExperience(employerAddress,
workDescription);
            await tx.wait(); // Wait for the transaction to be mined
            alert('Proposal has been sent!'); // Notify the user of success
        } catch (error) {
            console.error(error); // Log the error for debugging
            alert('Error in sending proposal. Please try again. Make sure you are
logged in using your employee metamask account.'); // Notify the user of the error
        }
    };

    return (
        <StyledContainer maxWidth="sm">
            <StyledCard elevation={16} sx={{ width: '400px', margin: 'auto',
padding: 5 }}>
                <Box
                    component="form"
                    sx={{ '& .MuiTextField-root': { m: 1, width: '100%' } }}
                    noValidate
                    autoComplete="off"
                    onSubmit={handleSubmit}
                >
                    <Typography component="h1" variant="h5" align="center"
color='primary' sx={{ fontWeight: 600 }}>
                        Propose Your Work Experience
                    </Typography>
                    {/* Text field for entering the employer address */}
                    <TextField
                        id="outlined-employer-address"
                        label="Employer Address"
                        name="employerAddress"
                        placeholder="Enter the employer account address"
                        multiline
                        rows={1}
                        value={employerAddress}
                        onChange={(e) => setEmployerAddress(e.target.value)}
```

```
                            fullWidth
                            InputProps={{
                                sx: {
                                    fontSize: '0.6rem',
                                },
                            }}
                        />
                        {/* Text field for entering work description */}
                        <TextField
                            id="outlined-work-description"
                            label="Work Experience Description"
                            name="workDescription"
                            placeholder="Propose a work experience description"
                            multiline
                            rows={5}
                            value={workDescription}
                            onChange={(e) => setWorkDescription(e.target.value)}
                            fullWidth
                        />
                        {/* Submit button */}
                        <Box sx={{ m: 1, display: 'flex', justifyContent: 'center' }}>
                            <Button type="submit" variant="contained">
                                Submit Proposal
                            </Button>
                        </Box>
                    </Box>
                </StyledCard>
            </StyledContainer>
        );
    }
```

Show Records

Frontend

- src/components/ShowRecords.js

```
  // State variables to manage checked items, records, user role, and alert
messages
  const [checked, setChecked] = useState([]); // Tracks which records are selected
for approval
  const [records, setRecords] = useState([]); // Stores the retrieved records
  const [isEmployer, setIsEmployer] = useState(null); // Indicates if the user is
an employer
  const [alertMessage, setAlertMessage] = useState(''); // Message for alerts

  // Toggles the checked state of a record when clicked
  const handleToggle = (value) => () => {
      const currentIndex = checked.indexOf(value);
      const newChecked = [...checked];
```

```
        // Add or remove the record from the checked list
        if (currentIndex === -1) {
            newChecked.push(value);
        } else {
            newChecked.splice(currentIndex, 1);
        }
        setChecked(newChecked); // Update the checked state
    };

    // Checks the user's role (employer or employee) to determine which records to
retrieve
    const checkRole = async () => {
        const contract = await connectEthereum();

        try {
            const role = await contract.checkRole(); // Call contract to check user
role
            if (role === 'employer') {
                setIsEmployer(true); // Set state if user is an employer
            } else if (role === 'employee') {
                setIsEmployer(false); // Set state if user is an employee
            } else {
                setAlertMessage('Register first'); // Alert if user is not
registered
            }
        } catch (error) {
            setAlertMessage('Network error'); // Alert on network errors
        }
    };

    // Retrieves records based on the user's role
    const retrieveRecords = async () => {

        const contract = await connectEthereum();
        const myAccounts = await window.ethereum.request({ method:
'eth_requestAccounts' });
        try {
            // Fetch records based on user role
            if (isEmployer) {
                const result = await contract.getEmployerRecords(myAccounts[0]);
                setRecords(result); // Store employer records
            } else {
                const result = await contract.getEmployeeRecords(myAccounts[0]);
                setRecords(result); // Store employee records
            }
        } catch (error) {
            setAlertMessage('No records'); // Alert if no records are found
        }
    };

    // Approves selected records by calling the contract method
    const approveRecords = async () => {
        const contract = await connectEthereum();
        let approvalList = [];
```

```
      // Gather records that are checked for approval
      for (const index of checked) {
        approvalList.push(records[index]);
        console.log(records[index]);
      }

      try{
          // Confirm each selected record
          for (const record of approvalList) {
              await contract.confirmWorkExperience(record.entryId); // Confirm
work experience in the contract
          }
      }catch(error){
          alert("rejected"); // alert for rejection
      }
    };
```

- `checkRole`: call `checkRole` in solidity to check if the current account is *employee* or *employer*, display correspondingly.
- `retrieveRecords`: use the current account address to call `getEmployerRecords` or `getEmployeeRecords` in solidity depending on the result form `checkRole`, the result retrieved in `retrieveRecords` will be displayed in this page. relevant to *employer* only:
- `handleToggle`: record down the checked entry for sending approval.
- `approveRecords`: call `confirmWorkExperience` in solidity for approving all checked record entry .

Backend

```
    // Function for the employer to confirm a work experience entry
    function confirmWorkExperience(uint256 entryId) external {
        WorkExperienceEntry storage entry = universalExperienceRecord[entryId];
        require(entry.employer != address(0), "This is not a valid entry");
        require(entry.employer == msg.sender, "Only the employer can confirm this
entry");
        require(!entry.confirmed, "Entry already confirmed");

        entry.confirmed = true;
        employers[msg.sender].reputationPoints += 1; // Increase reputation points
for confirmation

        emit WorkExperienceConfirmed(entryId, msg.sender);
    }

    // Function to get employee records
    function getEmployeeRecords(address employeeAddr) external view returns
(WorkExperienceEntry[] memory) {
        require(employees[employeeAddr].addr != address(0), "This is not a valid
employee account");

        uint256[] memory experienceIds =
employees[employeeAddr].submittedExperienceIds;
        WorkExperienceEntry[] memory records = new WorkExperienceEntry[]
(experienceIds.length);

        for (uint256 i = 0; i < experienceIds.length; i++) {
            records[i] = universalExperienceRecord[experienceIds[i]];
        }

        return records;
    }

    // Function to get employer records for their employees' experiences
    function getEmployerRecords(address employerAddr) external view returns
(WorkExperienceEntry[] memory) {
        require(employers[employerAddr].addr != address(0), "This is not a valid
employer account");

        uint256[] memory recordIds =
employers[employerAddr].receivedExperienceIds;
        WorkExperienceEntry[] memory records = new WorkExperienceEntry[]
(recordIds.length);

        for (uint256 i = 0; i < recordIds.length; i++) {
            records[i] = universalExperienceRecord[recordIds[i]];
        }

        return records;
    }
```

- `getEmployeeRecords` and `getEmployerRecords`:
  - check if the sender is valid.
  - retrive the entry ids linked to the account (`submittedExperienceIds` and `receivedExperienceIds`).
  - use the ids retrieved to get back the entries from `mapping universalExperienceRecord`.
  - return the list of entries.
- `confirmWorkExperience`:
  - retrive the entry from `mapping universalExperienceRecord` using the entryId in argument.
  - check if the sender and the entries are valid.
  - change the `comfirmed` in the entry to `true`
  - increment the `reputationPoints` of the *employer* (the one who approved the record)

Full frontend code

```
/**
 * ShowReceivedRecords Component
 *
 * This component displays a list of work experience records that have been
 * received by the logged-in user, allowing employers to review and approve
 * these records.
 *
 * Functionalities:
 * - Checks the user's role (employer or employee) upon mounting.
 * - Retrieves records based on the user's role:
 *   - Employers can view received work records.
 *   - Employees can view submitted work records (submitted in 'propose work
experience').
 * - Allows employers to approve selected records using checkboxes.
 * - Displays relevant information for each record, including employer and
 *   employee names, confirmation status, and additional details in a tooltip.
 *
 * State Management:
 * - Uses local state to manage the list of records, checked items for approval,
 *   the user's role, and any alert messages.
 *
 * User Interaction:
 * - Hover over comment icon to view detail description
 * - Users can toggle checkboxes to select records for approval.
 * - Provides visual feedback on the confirmation status of each record.
 * - Alerts the user in case of network errors or if MetaMask is not installed.
 *
 * The component is styled using Material-UI for a modern and responsive design.
 */

const ShowRecords = () => {
  // State variables to manage checked items, records, user role, and alert
messages
  const [checked, setChecked] = useState([]); // Tracks which records are selected
for approval
  const [records, setRecords] = useState([]); // Stores the retrieved records
  const [isEmployer, setIsEmployer] = useState(null); // Indicates if the user is
an employer
  const [alertMessage, setAlertMessage] = useState(''); // Message for alerts

  // Toggles the checked state of a record when clicked
  const handleToggle = (value) => () => {
      const currentIndex = checked.indexOf(value);
      const newChecked = [...checked];

      // Add or remove the record from the checked list
      if (currentIndex === -1) {
          newChecked.push(value);
      } else {
          newChecked.splice(currentIndex, 1);
      }
      setChecked(newChecked); // Update the checked state
  };

  // Checks the user's role (employer or employee) to determine which records to
retrieve
  const checkRole = async () => {
      const contract = await connectEthereum();

      try {
          const role = await contract.checkRole(); // Call contract to check user
role
          if (role === 'employer') {
              setIsEmployer(true); // Set state if user is an employer
          } else if (role === 'employee') {
              setIsEmployer(false); // Set state if user is an employee
          } else {
              setAlertMessage('Register first'); // Alert if user is not
registered
          }
      } catch (error) {
          setAlertMessage('Network error'); // Alert on network errors
      }
  };

  // Retrieves records based on the user's role
  const retrieveRecords = async () => {

      const contract = await connectEthereum();
      const myAccounts = await window.ethereum.request({ method:
'eth_requestAccounts' });
      try {
          // Fetch records based on user role
          if (isEmployer) {
              const result = await contract.getEmployerRecords(myAccounts[0]);
              setRecords(result); // Store employer records
          } else {
              const result = await contract.getEmployeeRecords(myAccounts[0]);
              setRecords(result); // Store employee records
          }
      } catch (error) {
          setAlertMessage('No records'); // Alert if no records are found
      }
  };

  // Approves selected records by calling the contract method
  const approveRecords = async () => {
      const contract = await connectEthereum();
      let approvalList = [];

      // Gather records that are checked for approval
      for (const index of checked) {
          approvalList.push(records[index]);
          console.log(records[index]);
        }

        try{
            // Confirm each selected record
            for (const record of approvalList) {
                await contract.confirmWorkExperience(record.entryId); // Confirm
work experience in the contract
            }
        }catch(error){
            alert("rejected"); // alert for rejection
        }
```

```
    };

    // Effect to check user role on component mount
    useEffect(() => {
        checkRole(); // Call checkRole function to determine user role
    }, []);

    // Effect to retrieve records when the user's role is determined
    useEffect(() => {
        if (isEmployer != null) {
            retrieveRecords(); // Fetch records based on user role
        }
    }, [retrieveRecords, isEmployer]);

  return (
    <StyledContainer>
        <StyledCard elevation={16} sx={{ width: '400px', margin: 'auto', padding:
5 }}>
            <Typography component="h1" variant="h5" align="center" color='primary'
sx={{ fontWeight: 600 }}>
                Records
            </Typography>
            <h1>{alertMessage}</h1> {/* Display alert messages to the user */}
            <List sx={{ width: '100%', maxWidth: 360, bgcolor: 'background.paper'
}}>
                {records.map((record, index) => {
                    const labelId = `checkbox-list-label-${index}`;

                    return (
                        <ListItem key={index} disablePadding>
                            {/* Button to toggle selection of the record */}
                            <ListItemButton role={undefined} onClick=
{handleToggle(index)} dense>
                                {isEmployer === true && !record.confirmed && (
                                    <ListItemIcon>
                                        <Checkbox
                                            edge="start"
                                            checked={checked.includes(index)} //
Check if the record is selected
                                            tabIndex={-1}
                                            disableRipple
                                            inputProps={{ 'aria-labelledby':
labelId }}
                                        />
                                    </ListItemIcon>
                                )}
                                </ListItemButton>
                                {/* Brief info of the record */}
                                <ListItemText
                                    id={labelId}
                                    secondary={
                                        <>
                                            <div style={{ color: '#003366',
fontWeight: 'bold' }}>
```

```
                                                        Employer - {record.employerName}
</div>
                                                    <div style={{ color: '#666666',
fontSize: '0.6rem' }}>
                                                        {record.employer}
                                                    </div>
                                                    <div style={{ color: '#4A90E2',
fontWeight: 'bold' }}>
                                                        Employee - {record.employeeName}
                                                    </div>
                                                    <div style={{ color: '#666666',
fontSize: '0.6rem' }}>
                                                        {record.employee}
                                                    </div>
                                                    <div style={{ color: record.confirmed
? '#28a745' : '#dc3545' }}>
                                                        Confirmed: {record.confirmed ?
'Yes' : 'No'}
                                                    </div> {/* Green for Yes, red for No
*/}
                                                </>
                                            }
                                        />
                                    {/* Tooltip to show the detail record description */}
                                    <Tooltip title={record.description} arrow>
                                        <IconButton edge="end" aria-label="comments">
                                            <CommentIcon />
                                        </IconButton>
                                    </Tooltip>
                                </ListItem>
                            );
                        })}
                    {/* Button for approving checked records, available for employer only
*/}
                    </List>
                    {isEmployer === true && (
                        <StyledButton
                            type="button"
                            variant="contained"
                            color="primary"
                            onClick={approveRecords}
                        >
                            Approve
                        </StyledButton>
                    )}
                </StyledCard>
            </StyledContainer>
        );
    }
```

EmployeePage

Frontend

- src/components/EmployeePage.js

```
   // State variables to manage records, alert messages, and employee address
input
   const [records, setRecords] = useState([]); // Stores the fetched work
experience records
   const [alertMessage, setAlertMessage] = useState(''); // Stores alert messages
for user feedback
   const [employeeAddress, setEmployeeAddress] = useState(''); // Stores the
employee's Ethereum address

   // Function to retrieve records from the blockchain based on the employee's
address
   const retrieveRecords = async () => {
      // Connect to the Ethereum contract
      const contract = await connectEthereum();

      try {
         // Fetch employee records using the provided address
         const result = await contract.getEmployeeRecords(employeeAddress);
         console.log(result); // Log the retrieved records for debugging
         setRecords(result); // Update the state with the fetched records
         setAlertMessage(''); // Clear any previous alert messages
      } catch (error) {
         console.error(error); // Log error for debugging
         setRecords([]); // Reset records in case of error
         setAlertMessage('No records found'); // Alert if no records are found
      }
   };
```

- `employeeAddress`: retrived by text field
- `retrieveRecords`: call `getEmployeeRecords` with the `employeeAddress`, the result set the `records`, which is then displayed in the page

Backend

- `getEmployeeRecords`: Explained in **Show Records**

Full frontend code

```
/**
 * EmployeePage Component
 *
 * This component provides a user interface for employees to search for and view
 * their work experience records stored on a blockchain. Users can input their
 * Ethereum address to retrieve associated records, including details about
 * their employers and confirmation status.
 *
 * Main Features:
 * - Allows users to enter their Ethereum account address to fetch their work
 *   experience records.
 * - Displays a list of records with relevant details, including employer names,
 *   addresses, and confirmation status.
 * - Utilizes Material-UI components for a modern and responsive design.
 * - Provides user feedback through alerts, such as when no records are found
 *   or when MetaMask is not installed.
 *
 * State Management:
 * - Uses React state to manage the list of records, alert messages, and the
 *   employee's address input.
 *
 * User Interaction:
 * - Users can input their Ethereum address in a text field, triggering a
 *   retrieval of records upon blur (losing focus).
 * - Each record is displayed in a list format, with tooltips providing
 *   additional information about each record.
 *
 */
const EmployeePage = () => {
    // State variables to manage records, alert messages, and employee address
input
    const [records, setRecords] = useState([]); // Stores the fetched work
experience records
    const [alertMessage, setAlertMessage] = useState(''); // Stores alert messages
for user feedback
    const [employeeAddress, setEmployeeAddress] = useState(''); // Stores the
employee's Ethereum address

    // Function to retrieve records from the blockchain based on the employee's
address
    const retrieveRecords = async () => {
        // Connect to the Ethereum contract
        const contract = await connectEthereum();

        try {
            // Fetch employee records using the provided address
            const result = await contract.getEmployeeRecords(employeeAddress);
            console.log(result); // Log the retrieved records for debugging
            setRecords(result); // Update the state with the fetched records
            setAlertMessage(''); // Clear any previous alert messages
        } catch (error) {
            console.error(error); // Log error for debugging
            setRecords([]); // Reset records in case of error
            setAlertMessage('No records found'); // Alert if no records are found
        }
    };

    // Effect to retrieve records whenever the employee address changes
    useEffect(() => {
        if (employeeAddress) {
            retrieveRecords(); // Call retrieveRecords if an address is set
        }
    }, [retrieveRecords, employeeAddress]);

    return (
        <StyledContainer>
            <StyledCard elevation={16} sx={{ width: '400px', margin: 'auto',
padding: 5 }}>
                <Typography component="h1" variant="h5" align="center"
color='primary' sx={{ fontWeight: 600 }}>
                    Search Employee
                </Typography>
                {/* Input field for employee's Ethereum address */}
                <TextField
                    id="outlined-employee-address"
                    label="Employee Address"
                    placeholder="Enter the employee account address"
                    onBlur={(e) => setEmployeeAddress(e.target.value.trim())} //
Update state on blur
                    fullWidth
                    InputProps={{
                        sx: {
                            fontSize: '0.6rem', // Set font size for input
                        },
                    }}
                />
                <h1>{alertMessage}</h1> {/* Display alert messages */}
                <Typography component="h1" variant="h5" align="center" sx={{
fontWeight: 600, color: 'purple' }}>
                    {records.length > 0 ? `Records` : ''} {/* Show "Records"
header if any records are found */}
                </Typography>

                <List sx={{ width: '100%', maxWidth: 360, bgcolor:
'background.paper' }}>
                    {/* Map through the records and display them in a list */}
                    {records.map((record, index) => (
                        record.confirmed && (
                        <ListItem key={index} disablePadding>
                            <ListItemText
                                secondary={
                                        <>
                                            <div style={{ color: '#003366',
fontWeight: 'bold' }}>
                                                Employer - {record.employerName}
{/* Display employer name */}
                                            </div>
                                            <div style={{ color: '#666666',
fontSize: '0.6rem' }}>
                                                {record.employer} {/* Display
employer address */}
                                            </div>
                                            <div style={{ color: record.confirmed
? '#28a745' : '#dc3545' }}>
                                                Confirmed: {record.confirmed ?
'Yes' : 'No'} {/* Display confirmation status */}
                                            </div>
                                        </>
                                }
                                />
                            {/* Tooltip for showing additional description of the
record */}
                            <Tooltip title={record.description} arrow>
                                <IconButton edge="end" aria-label="comments">
                                    <CommentIcon /> {/* Comment icon for
additional details */}
                                </IconButton>
                            </Tooltip>

                        </ListItem>
                    )))}
                </List>
            </StyledCard>
        </StyledContainer>
    );
}
```

EmployerPage

Frontend

- src/components/EmployerPage.js

```

   // State variables to manage employer list, mode, selected employer, peer
comments, votes, and comments
   const [employerList, setEmployerList] = useState([]); // List of employers
   const [mode, setMode] = useState(null); // Current mode (selecting or
commenting)
   const [selectedEmployer, setSelectedEmployer] = useState(null); // Currently
selected employer
   const [peerComments, setPeerComments] = useState([]); // Comments from peers
   const [vote, setVote] = useState(null); // Vote state (true for upVote, false
for downVote)
   const [commentText, setCommentText] = useState(); // Text for the comment input
   const [isEmployer, setIsEmployer] = useState(null); // Indicates if the user is
an employer

   // Modes for the component
   const MODES = {
      SELECTING: 1,
      COMMENTING: 2
   };

   // Load the employer list from the blockchain
   const loadEmployerList = async (page) => {
      try {
            const contract = await connectEthereum();
            const result = await contract.getEmployerList(page);
            setEmployerList(result); // Set the loaded employer list
        } catch (error) {
            console.error("Error loading employer list:", error);
        }
    };

    // Fetch peer comments for a selected employer
    const fetchPeerComments = async (addr) => {
        try {
            const contract = await connectEthereum();
            const result = await contract.getPeerComments(addr);
            setPeerComments(result); // Set the fetched comments for the employer
        } catch (error) {
            alert("Error fetching peer comments");
        }
    };

    // Handle the selection of an employer
    const handleSelectEmployer = (employer) => {
        setMode(MODES.SELECTING); // Switch to selecting mode
        setSelectedEmployer(employer); // Set the selected employer
        fetchPeerComments(employer.addr); // Fetch comments for the selected
employer
    };

    // Switch to commenting mode
    const makeComment = () => {
        setMode(MODES.COMMENTING);
    };

    // Handle changes in voting (upVote or downVote)
    const handleVoteChange = (event) => {
        if (event.target.value === 'downVote') {
            setVote(false); // Set vote to downVote
        } else {
            setVote(true); // Set vote to upVote
        }
    };

    // Send the comment to the blockchain
    const sendComment = async () => {
        try {
            const contract = await connectEthereum();
            await contract.peerComment(commentText, vote, selectedEmployer.addr);
            alert('Comment sent'); // Notify user of successful comment
        } catch (error) {
            alert("Rejected. (You can only make as many comments as your reputation
points)");
        }
    };

    // Check if the current user is an employer
    const checkIsEmployer = async () => {
      try {
        const contract = await connectEthereum();
        const role = await contract.checkRole();
        setIsEmployer(role === 'employer'); // Set employer status based on role
      } catch (error) {
        console.error("Error checking role:", error);
        setIsEmployer(false); // Default to not employer on error
      }
    };
```

- General control:
  - `checkIsEmployer`: call `checkRole` in solidity to check if the account is *employer*, if it is, display the **COMMENT** button with the corresponding comment function.
  - `mode` and `MODES`: control whether the right panel should display a list of peer comments (in mode `SELECTING`) or the comment text field (in mode `COMMENTING`).
- For displaying employer list in right panel:
  - `loadEmployerList`: call `getEmployerList` in solidity to retrieve a list of *employer* with `name`, `addr`, `upvote`, `downvote`, `reputationPoints`.
- For displaying peer comment in left panel:
  - `handleSelectEmployer`:
    - call when click on one of the *employer* on the left panel.
    - change state `mode` to (`SELECTING`).
    - call `fetchPeerComment`.
  - `fetchPeerComment`: call `getPeerComments` and set `peerComments` for displaying a list of peer comments on the right panel.
- For making comments in right panel (available to *employer* only):
  - `makeComment`: change state `mode` to (`COMMENTING`) and display the comment text field.
  - `handleVoteChange`: record if the user select **Up Vote** or **Down Vote** in state `vote`.
  - `sendComment`: call `peerComment` to make peer comment.

Backend

```
    struct PeerComment {
        bool upVote;
        string comment;
        string commentorName;
        address commentorAddress;
    }

    mapping(address => Employer) internal employers; // stores all employers
    mapping(uint256 => PeerComment) internal peerCommentRecord; // stores all
peerComment
    address[] internal employerAddresses; // store employer address
    uint256 internal nextPeerCommentId; // Track next PeerComment ID

    // Function to get a paginated list of employers, page start from 1
    function getEmployerList(uint256 page) external view returns (Employer[]
memory) {
        uint256 pageSize = 10; // Define how many employers per page
        uint256 startIndex = (page - 1) * pageSize;
        uint256 endIndex = startIndex + pageSize > employerAddresses.length ?
employerAddresses.length : startIndex + pageSize;

        require(startIndex < employerAddresses.length, "Page index out of
bounds");

        Employer[] memory employerList = new Employer[](endIndex - startIndex);

        for (uint256 i = startIndex; i < endIndex; i++) {
            employerList[i - startIndex] = employers[employerAddresses[i]];
        }

        return employerList;
    }

    // Function that allows an employer to comment on another employer
    function peerComment(string calldata commentText, bool upVote, address to)
external {
        require(employers[msg.sender].addr != address(0), "This is not a valid
employer account");
        require(employers[msg.sender].commentMadeCounter <
employers[msg.sender].reputationPoints,
                "You have made more comments than your reputation points");
        require(to != msg.sender, "You cannot comment on yourself"); // Prevent
self-commenting

        PeerComment memory newComment = PeerComment({
            upVote: upVote,
            comment: commentText,
            commentorName: employers[msg.sender].name,
            commentorAddress: msg.sender
        });

        peerCommentRecord[nextPeerCommentId] = newComment;
        employers[to].peerCommentIds.push(nextPeerCommentId);
        nextPeerCommentId++; // Increment the peerCommentId
        employers[msg.sender].commentMadeCounter += 1; // Increment the comment
counter

        if (upVote) {
            employers[to].upVote += 1;
        } else {
            employers[to].downVote += 1;
        }

        emit PeerCommentAdded(to, msg.sender, commentText, upVote);
    }

    // Function to get comments made to an employer
    function getPeerComments(address employerAddr) external view returns
(PeerComment[] memory) {
        uint256[] memory commentIds = employers[employerAddr].peerCommentIds;
        PeerComment[] memory comments = new PeerComment[](commentIds.length);

        for (uint256 i = 0; i < commentIds.length; i++) {
            comments[i] = peerCommentRecord[commentIds[i]];
        }

        return comments;
    }
```

- `getEmployerList`:
  - `[] employerAddresses` stores address of all *employers*.
  - address retreived from `[] employerAddresses` is then used to retrieve *employer* from `mapping employers` one by one.
  - return the *employers* retrieved for display of employer list.
- `peerComment`:
  - check if the commentator and the comment target is valid.
  - check if the `commentMadeCounter`(number of comment made by the commentator) exceeds `reputationPoints`.
  - new `PeerComment` with its field set accourding to the arguments.
  - store in `mapping peerCommentRecord`, the commentator stores the entry id `nextPeerCommentId` in the `[] peerCommentIds` as a reference, then increment `nextPeerCommentId`.
  - increment the commentator's `commentMadeCounter` to record the number of comment made.
- `getPeerComments`:
  - use the employer address `employerAddr` in the argument to retrieve the list of `[] peerCommentIds` of that *employee*
  - use the entry id to retrieve peer comments from `mapping peerCommentRecord` one by one, and return.

Full frontend code

```
/**
 * EmployerPage Component
 *
 * This component serves as a user interface for interacting with a list of
employers
 * on a blockchain platform. It allows users to view employer details, including
peer
 * comments and voting options (upVote or downVote). The component is designed to
handle
 * both employer and employee roles, enabling employers to leave comments and vote
on
 * other employers.
 *
 * Main Features:
 * - Displays a list of employers retrieved from the blockchain.
 * - Users can select an employer to view detailed information, including their
```

```
reputation
 *   points, upVote/downVote counts, and peer comments.
 * - Provides functionality to leave comments on selected employers, with the
option to
 *   upVote or downVote.
 * - Checks the user's role (employer or employee) to determine available actions.
 *
 * State Management:
 * - Uses React state to manage the employer list, the current mode (selecting or
commenting),
 *   the selected employer, peer comments, the user's vote choice, and the comment
text input.
 *
 * User Interaction:
 * - Users can click on an employer to see their details and peer comments.
 * - Employers can leave comments and vote on the employers they select.
 * - Alerts are provided for errors, such as when fetching data or sending
comments.
 *
 * The layout is built using Material-UI components for a responsive and modern
design,
 * ensuring a seamless user experience across devices.
 */

// Handle randering of employer list
const EmployerList = ({ employerList, onSelect }) => (
    <List>
        {employerList.map((employer, index) => (
            <ListItem button key={index} onClick={() => onSelect(employer)}>
                <Box sx={{ display: 'flex', flexDirection: 'column', width: '100%'
}}>
                    <Box sx={{ display: 'flex', flexDirection: 'row' }}>
                        <ListItemText primary={employer.name} sx={{ color:
'purple' }} />
                        {/* Reputation points display */}
                        <Box
                            bgcolor="primary.main"
                            color="white"
                            px={0.5}
                            borderRadius="50%"
                            fontSize="small"
                            display="flex"
                            justifyContent="center"
                            alignItems="center"
                            width={20}
                            height={20}
                            sx={{ margin: '0.3rem', marginRight: 8 }}
                        >
                            {employer.reputationPoints.toString()}
                        </Box>
                        {/* Up vote display */}
                        <Box sx={{ color: 'green', display: 'flex', flexDirection:
'row' }}>
                            <Icon><ArrowUpward /></Icon>
```

```
                                    <Typography>{employer.upVote.toString()}</Typography>
                                </Box>
                                {/* Down vote display */}
                                <Box sx={{ color: 'red', display: 'flex', flexDirection:
'row' }}>
                                    <Icon><ArrowDownward /></Icon>
                                    <Typography>{employer.downVote.toString()}
</Typography>
                                </Box>
                            </Box>
                            {/* Employer address display */}
                            <Typography sx={{ fontSize: '0.6rem', color: 'text.secondary'
}}>
                                {employer.addr}
                            </Typography>
                        </Box>
                    </ListItem>
                ))}
            </List>
);

const EmployerPage = () => {
    // State variables to manage employer list, mode, selected employer, peer
comments, votes, and comments
    const [employerList, setEmployerList] = useState([]); // List of employers
    const [mode, setMode] = useState(null); // Current mode (selecting or
commenting)
    const [selectedEmployer, setSelectedEmployer] = useState(null); // Currently
selected employer
    const [peerComments, setPeerComments] = useState([]); // Comments from peers
    const [vote, setVote] = useState(null); // Vote state (true for upVote, false
for downVote)
    const [commentText, setCommentText] = useState(); // Text for the comment
input
    const [isEmployer, setIsEmployer] = useState(null); // Indicates if the user
is an employer

    // Modes for the component
    const MODES = {
        SELECTING: 1,
        COMMENTING: 2
    };

    // Load the employer list from the blockchain
    const loadEmployerList = async (page) => {
        try {
            const contract = await connectEthereum();
            const result = await contract.getEmployerList(page);
            setEmployerList(result); // Set the loaded employer list
        } catch (error) {
            console.error("Error loading employer list:", error);
        }
    };
    // Fetch peer comments for a selected employer
    const fetchPeerComments = async (addr) => {
        try {
            const contract = await connectEthereum();
            const result = await contract.getPeerComments(addr);
            setPeerComments(result); // Set the fetched comments for the employer
        } catch (error) {
            alert("Error fetching peer comments");
        }
    };

    // Handle the selection of an employer
    const handleSelectEmployer = (employer) => {
        setMode(MODES.SELECTING); // Switch to selecting mode
        setSelectedEmployer(employer); // Set the selected employer
        fetchPeerComments(employer.addr); // Fetch comments for the selected
employer
    };

    // Switch to commenting mode
    const makeComment = () => {
        setMode(MODES.COMMENTING);
    };

    // Handle changes in voting (upVote or downVote)
    const handleVoteChange = (event) => {
        if (event.target.value === 'downVote') {
            setVote(false); // Set vote to downVote
        } else {
            setVote(true); // Set vote to upVote
        }
    };

    // Send the comment to the blockchain
    const sendComment = async () => {
        try {
            const contract = await connectEthereum();
            await contract.peerComment(commentText, vote, selectedEmployer.addr);
            alert('Comment sent'); // Notify user of successful comment
        } catch (error) {
            alert("Rejected. (You can only make as many comments as your
reputation points)");
        }
    };

    // Check if the current user is an employer
    const checkIsEmployer = async () => {
        try {
            const contract = await connectEthereum();
            const role = await contract.checkRole();
            setIsEmployer(role === 'employer'); // Set employer status based on
role
        } catch (error) {
            console.error("Error checking role:", error);
            setIsEmployer(false); // Default to not employer on error
        }
    };

    // Load employer list and check user role on component mount
    useEffect(() => {
        loadEmployerList(1);
        checkIsEmployer();
    }, [loadEmployerList, checkIsEmployer]);

    let content; // Variable to hold the content based on mode

    // Determine content based on the current mode
    if (mode === MODES.SELECTING) {
        content = (
            <>
                <Typography variant="h6" sx={{color: 'purple'}}>
{selectedEmployer?.name}</Typography>
                <Typography variant="subtitle1" sx={{fontSize: '0.6rem', color:
'text.secondary'}}>{selectedEmployer?.addr}</Typography>
                <Typography variant="h6" color='primary'>Peer Comments:
</Typography>
                {peerComments.length > 0 ? (
                    // if there are peer comments, display one by one
                    peerComments.map((comment, index) => (
                        <>
                        {/* Reputation points display */}
                        <Box sx={{ display: 'flex', flexDirection: 'row' }}>
                            <Typography sx={{color: 'purple'}} key={index}>{index
+ 1}. {comment.commentorName}</Typography>
                            {/* Vote display */}
                            {comment.upVote?(
                            // upvote icon
                            <Box sx={{ color: 'green', display: 'flex',
flexDirection: 'row' }}>
                                <Icon><ArrowUpward /></Icon>
                            </Box>)
                            :
                            // downvote icon
                            (<Box sx={{ color: 'red', display: 'flex',
flexDirection: 'row' }}>
                                <Icon><ArrowDownward /></Icon>
                            </Box>)
                            }
                        </Box>
                        {/* commentor address display */}
                        <Typography variant="subtitle1" sx={{fontSize: '0.6rem',
color: 'text.secondary'}}>{comment.commentorAddress}</Typography>
                        {/* peer comment display */}
                        <Typography key={index}>{comment.comment}</Typography>
                        </>
                    ))
                ) : (
                    // display if no peer comment
                    <Typography>No comments available</Typography>
                )}
                {isEmployer &&
                    <StyledButton
                        type="button"
                        variant="contained"
                        color="primary"
                        onClick={makeComment}
                    >
                        Comment
                    </StyledButton>
                }
            </>
        );
    } else if (mode === MODES.COMMENTING) {
        content = (
            <>
                <Typography variant="h6" sx={{ color: 'purple' }}>Make comment to
{selectedEmployer.name}</Typography>
                <Typography variant="subtitle1" sx={{ fontSize: '0.6rem', color:
'text.secondary' }}>{selectedEmployer?.addr}</Typography>
                <TextField
                    label="Comments"
                    multiline
                    rows={4}
                    variant="outlined"
                    fullWidth
                    onChange={(e) => setCommentText(e.target.value)} // Update
comment text on change
                    sx={{ margin: '16px 0' }}
                />
                <FormControl component="fieldset">
                    <RadioGroup
                        aria-label="voting"
                        name="voting"
                        value={vote === true ? 'upVote' : 'downVote'} // Set
current vote value
                        onChange={handleVoteChange} // Handle vote changes
                    >
                        <FormControlLabel
                            value="upVote"
                            control={<Radio />}
                            label="Up Vote"
                        />
                        <FormControlLabel
                            value="downVote"
                            control={<Radio />}
                            label="Down Vote"
                        />
                    </RadioGroup>
                </FormControl>
                <StyledButton
                    type="button"
                    variant="contained"
                    color="primary"
                    onClick={sendComment} // Send the comment to the blockchain
                >
                    Send Comment
                </StyledButton>
            </>
        );
    } else {
        content = <Typography variant="h6">Select an employer to see
details</Typography>; // Default content
    }

    return (
        <StyledContainer maxWidth="sm">
            <StyledCard elevation={16} sx={{ width: '400px', margin: 'auto',
padding: 5 }}>
                {/* Left Sidebar for displaying the list of employers */}
                <Paper sx={{ width: '250px', padding: 2 }}>
                    <Typography variant="h6" gutterBottom>Employers</Typography>
                    <EmployerList employerList={employerList} onSelect=
{handleSelectEmployer} /> {/* Render the employer list */}
                </Paper>
            </StyledCard>

            {/* Right Detail Area for displaying selected employer's details */}
            <StyledCard elevation={16} sx={{ width: '400px', margin: 'auto',
padding: 5 }}>
                <Box sx={{ flexGrow: 1, padding: 2 }}>
                    {content} {/* Render the content based on the current mode */}
                </Box>
            </StyledCard>
        </StyledContainer>
    );
};
```