
echolink-dapp Documentation

Release 0.0.1

Harsh Rathi

Jan 11, 2019

Contents:

1	Introduction	3
2	EkoLink DApp	5
2.1	Registration and Authentication	5
2.2	Dashboard	5
2.3	Upload Credentials	5
2.4	Query Credentials	9
2.4.1	View Credentials: User Profile	11
2.5	Post Job	11
2.6	Query Job	11
2.7	Post Experience	13
2.8	Query Experience	15
2.9	Bulk Upload Credentials	15
3	EKO Token Bridge	19
3.1	Steps to start the services	19
3.2	Steps to use the Eko Transfer service	20
3.3	Steps to use the Token Bridge service	20
3.4	Steps to use the Reverse Token Bridge service	20
4	Authority Management For EKO Platform	23
4.1	Steps to set up the EKO network and dynamically manage network authorities:	23
4.2	Using Docker to setup the nodes in the network	37
5	EKO Extension	39
5.1	How to use extension	39



EkoLink is a blockchain based system that provides verified education, skill, and work experience information. Taking advantage of blockchain technology's immutability and time stamp functionality, EkoLink provides users with trusted information regarding a job candidate's education, skill, and work experience. EkoLink is built on the EKO Blockchain Platform.

EKO Blockchain Platform is a public blockchain service built on top of Proof of Professional Stake consensus protocol (PoPS). PoPS is designed for enterprise grade blockchain applications. The EKO Blockchain Platform is fully compatible with EVM based Solidity smart contracts, and offers some innovative features, such as confidential contracts. The EKO Blockchain Platform provides businesses and developers a fast and efficient way to develop and deploy blockchain applications.

CHAPTER 1

Introduction

EkoLink is a blockchain based system that provides verified education, skill, and work experience information. Taking advantage of blockchain technology's immutability and time stamp functionality, EkoLink provides users with trusted information regarding a job candidate's education, skill, and work experience. EkoLink is built on the EKO Blockchain Platform.

EKO Blockchain Platform is a public blockchain service built on top of Proof of Professional Stake consensus protocol (PoPS). PoPS is designed for enterprise grade blockchain applications. The EKO Blockchain Platform is fully compatible with EVM based Solidity smart contracts, and offers some innovative features, such as confidential contracts. The EKO Blockchain Platform provides businesses and developers a fast and efficient way to develop and deploy blockchain applications.

EchoLink is a blockchain based system that provides verified education, skill, and work experience information. Taking advantage of blockchain technology's immutability and time stamp functionality, EchoLink provides users with trusted information regarding a job candidate's education, skill, and work experience. EkoLink is built on the EKO Blockchain Platform.

2.1 Registration and Authentication

When you browse to our [platform](#) you will be shown a login page like the one below.

You should have a private account with mnemonic key or `priv_key` to start a transaction. If you don't already have one, you create a new account by clicking the `create` button, this will show you the screen like the one below.

It is important that you retrieve your new mnemonic key somewhere safe, you can also download the key as the text file. Then you need to click on to `I've copied it somewhere safe` and enter your mnemonic.

The demo below shows how successful login using mnemonic leads to your dashboard

2.2 Dashboard

After successful authentication using your mnemonic, you will be redirected to your dashboard as shown below.

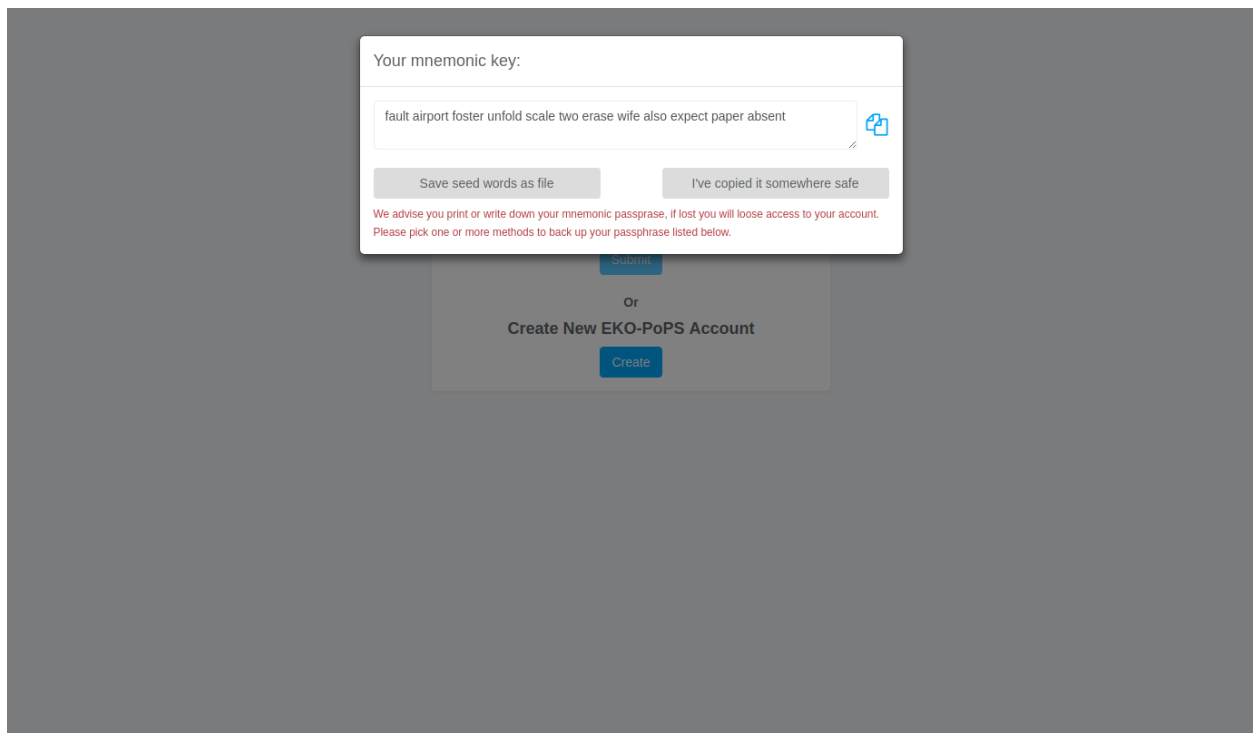
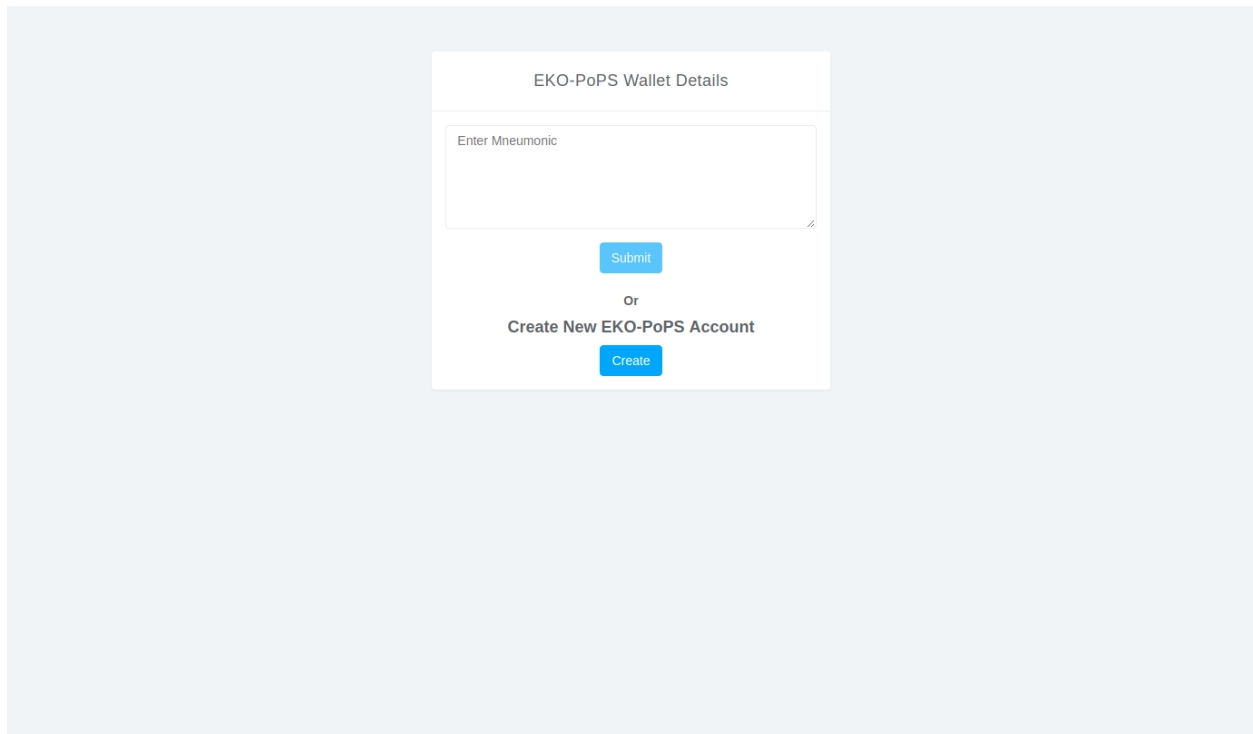
On the top right corner you can see the default address of your account and your EKO PoPS Energy as shown below.

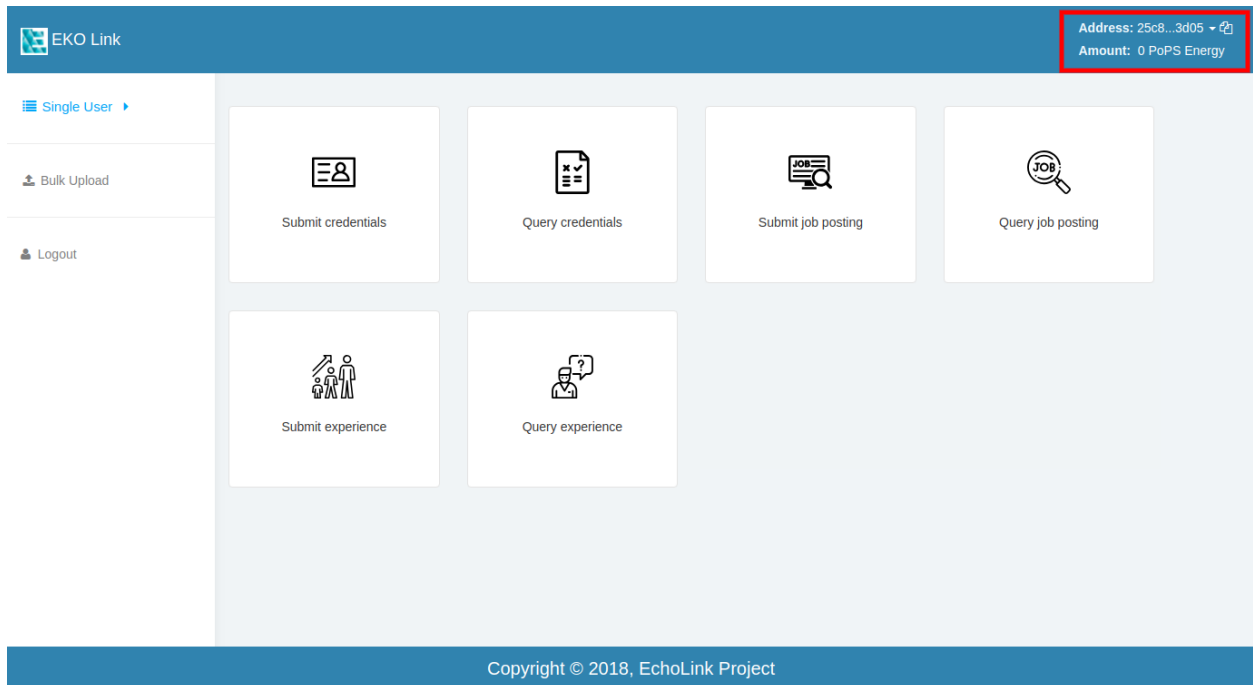
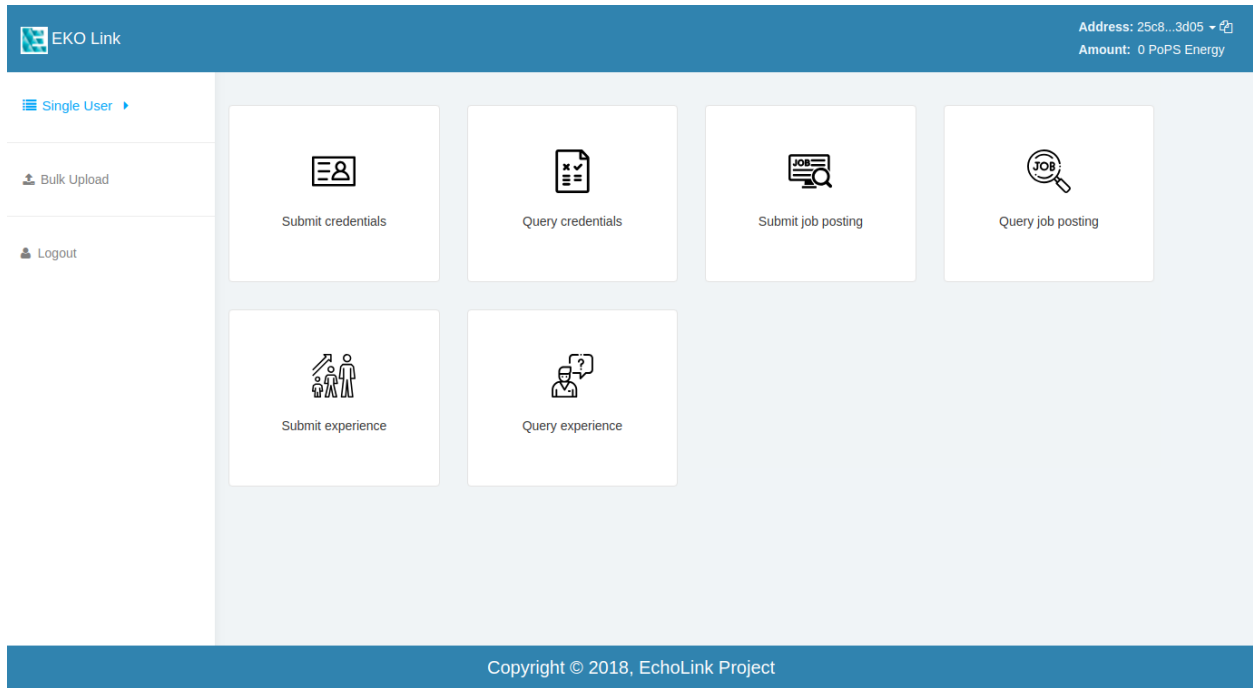
You can select upto 5 address with your mnemonic account.

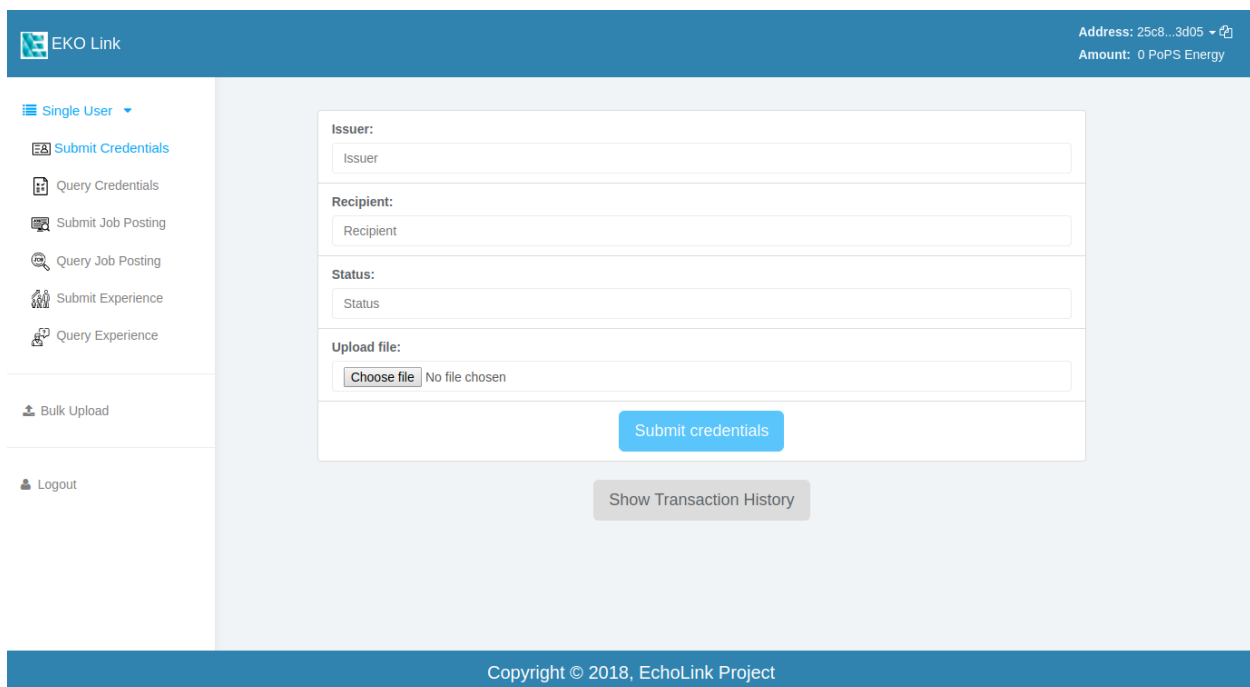
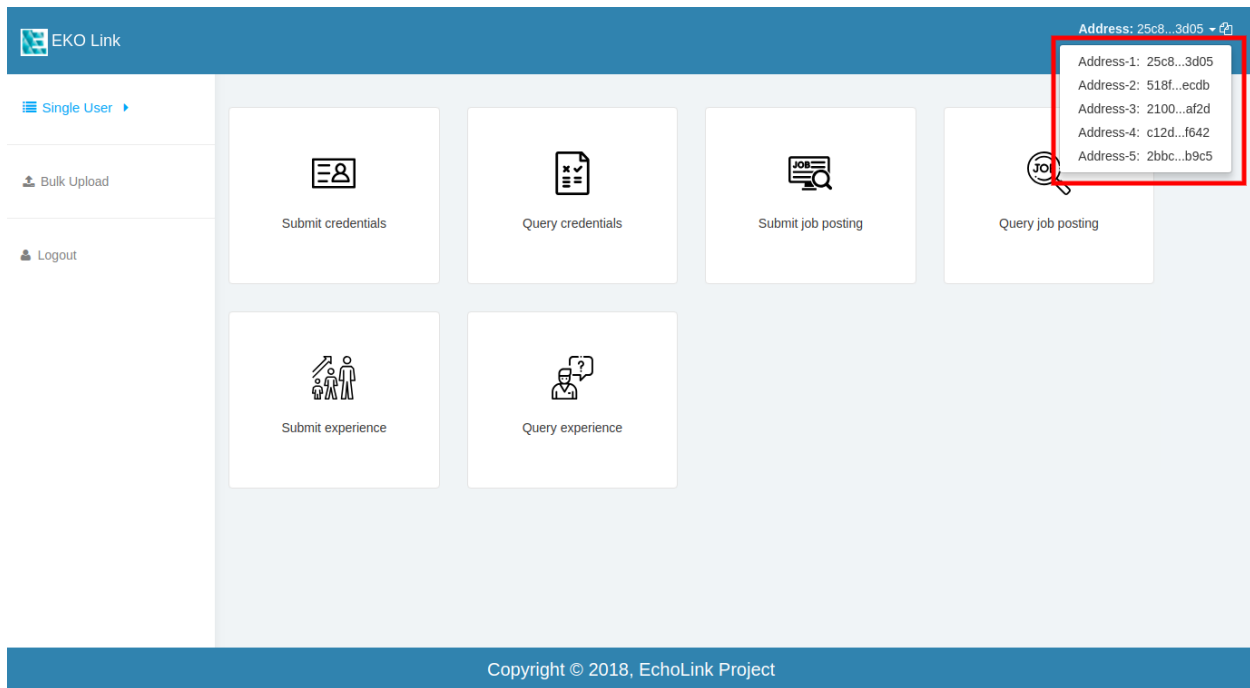
2.3 Upload Credentials

This is page where you can upload your credentials to our EKO-Pops Node.

Let's add some data and upload the credentials







The screenshot displays the EKO Link application interface. At the top, a blue header bar contains the 'EKO Link' logo on the left and the user's address '25c8...3d05' and amount '100 PoPS Energy' on the right. A left sidebar menu lists options: 'Single User' (selected), 'Submit Credentials', 'Query Credentials', 'Submit Job Posting', 'Query Job Posting', 'Submit Experience', 'Query Experience', 'Bulk Upload', and 'Logout'. The main content area features a form for submitting credentials with fields for 'Issuer' (Patrick Jane), 'Recipient' (Helen), 'Status' (Graduated), and an 'Upload file' section with a 'Choose file' button and '1.png' listed. A blue 'Submit credentials' button is at the bottom of the form, and a grey 'Show Transaction History' button is below it. The footer shows 'Copyright © 2018, EchoLink Project'.

Here you can see the demo of how the transaction hash is generated for our transaction on the network. You can retrieve the transaction hash and check the status on our node or the status of the transaction will be shown at the end of the transaction as you'll see below.

For a successful transaction, you need to have your account address approved by the admin and then have some token energy. You can see the token being energy updated at the top right corner with the successful transaction. Once you've done the transaction, you can see the history of the transactions with time and status by clicking on `Show Transactions history` button. If any transaction fails, you'll have an option to resubmit that transaction without having to enter the credentials again. And if the transaction is successful, you'll see the the updated time and status in the transaction history table.

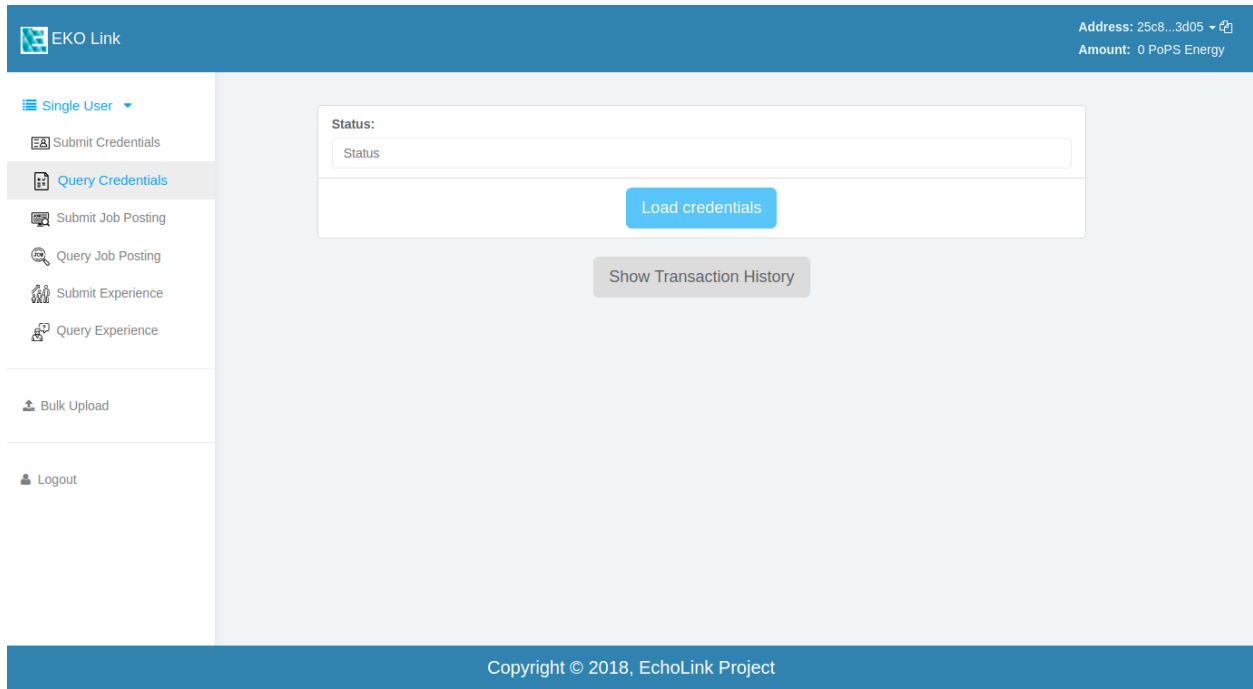
2.4 Query Credentials

In this section of our application you can query the credentials you've uploaded on our EKO-Pops Node. The image below shows the screenshot of the page.

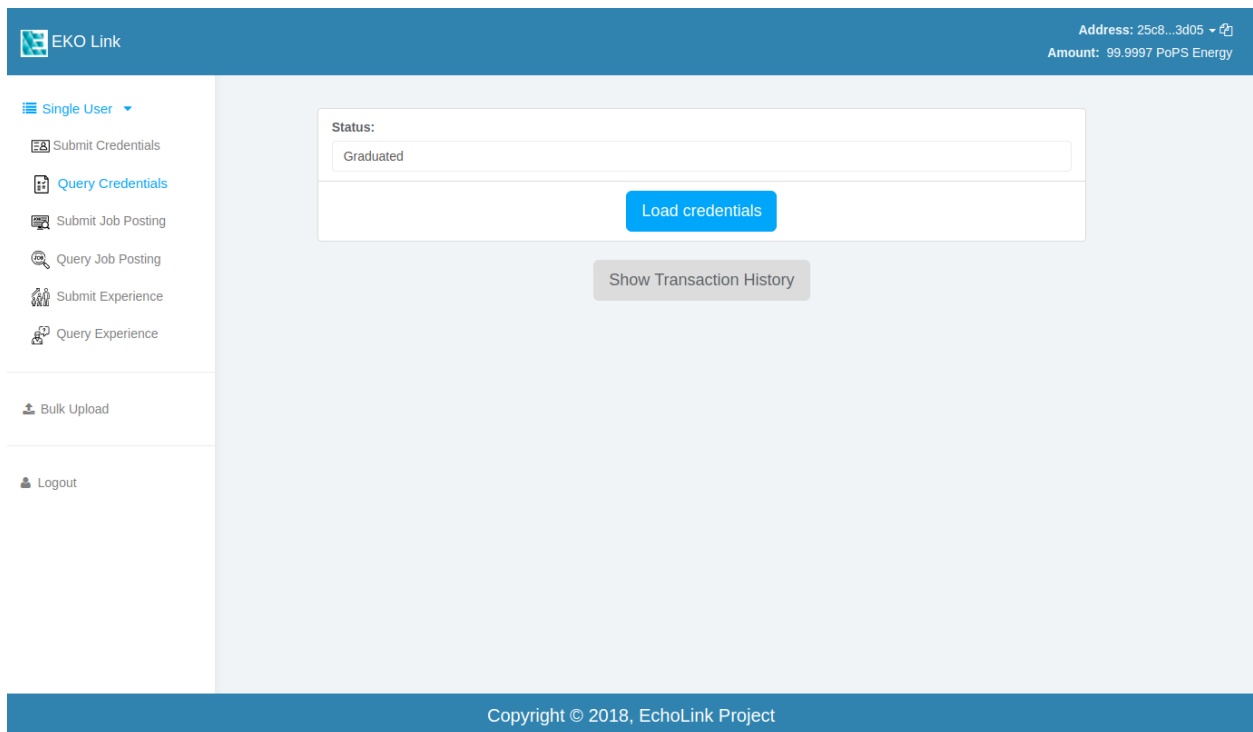
Let's query the credentials which we uploaded in our previous section on *upload credentials*.

Here you can see the demo of how you can retrieve your credentials, data will be displayed when the correct status is entered.

Here you can preview the image you've uploaded with your credentials and can download it in your local system. Also, Once you've done the transaction, you can see the history of the transactions with time and address by clicking on `Show Transactions history` button.



The screenshot shows the EKO Link DApp interface. The top header is blue with the EKO Link logo on the left and the user's address (25c8...3d05) and amount (0 PoPS Energy) on the right. A left sidebar contains a 'Single User' dropdown menu and several options: 'Submit Credentials', 'Query Credentials' (highlighted), 'Submit Job Posting', 'Query Job Posting', 'Submit Experience', 'Query Experience', 'Bulk Upload', and 'Logout'. The main content area has a 'Status:' label above a dropdown menu currently showing 'Status'. Below the dropdown is a blue 'Load credentials' button and a grey 'Show Transaction History' button. The footer is blue with the text 'Copyright © 2018, EchoLink Project'.



This screenshot is identical to the one above, showing the EKO Link DApp interface. The only difference is that the 'Status:' dropdown menu in the main content area now displays 'Graduated' instead of 'Status'. All other elements, including the header, sidebar, buttons, and footer, remain the same.

2.4.1 View Credentials: User Profile

You can see on the demo above, there's `view credential` option below the image you queried. Clicking on that will lead you to a new page which can serve as your public profile. You can access it anywhere without having to login.

The profile page will look like the one below.

2.5 Post Job

This is page where you can post a job to our EKO-Pops Node.

Let's add some data and post the job

Here you can see the demo of how the transaction hash is generated for our transaction on the network. You can retrieve the transaction hash and check the status on our node or the status of the transaction will be shown at the end of the transaction as you'll see below.

For a successful transaction, you need to have your account address approved by the admin and then have some token energy. You can see the token being energy updated at the top right corner with the successful transaction. Once you've done the transaction, you can see the history of the transactions with time and status by clicking on `Show Transactions history` button. If any transaction fails, you'll have an option to resubmit that transaction without having to enter the credentials again. And if the transaction is successful, you'll see the the updated time and status in the transaction history table.

2.6 Query Job

In this section of our application you can search for the job on our EKO-Pops Node. The image below shows the screenshot of the page.

EKO Link

Address: 25c8...3d05

Amount: 99.9997 PoPS Energy

Single User

Submit Credentials

Query Credentials

Submit Job Posting

Query Job Posting

Submit Experience

Query Experience

Bulk Upload

Logout

Job post:

NodeJS

Post Job

Show Transaction History

Copyright © 2018, EchoLink Project

EKO Link

Address: 25c8...3d05

Amount: 0 PoPS Energy

Single User

Submit Credentials

Query Credentials

Submit Job Posting

Query Job Posting

Submit Experience

Query Experience

Bulk Upload

Logout

Job post:

Post: Java, ReactJS, NodeJs, etc.

Load Job

Show Transaction History

Copyright © 2018, EchoLink Project

Let's find the job which we posted in our previous section on *post job*.

The screenshot shows the EKO Link web application interface. At the top, there's a blue header bar with the 'EKO Link' logo on the left and user information on the right: 'Address: 25c8...3d05' and 'Amount: 99.9995 PoPS Energy'. On the left side, there's a sidebar menu with options: 'Single User' (expanded), 'Submit Credentials', 'Query Credentials', 'Submit Job Posting', 'Query Job Posting' (highlighted), 'Submit Experience', 'Query Experience', 'Bulk Upload', and 'Logout'. The main content area features a 'Job post:' form with a text input field containing 'NodeJS' and a blue 'Load Job' button. Below the form is a grey button labeled 'Show Transaction History'. At the bottom of the page, a blue footer bar contains the text 'Copyright © 2018, EchoLink Project'.

Here you can see the demo of how you can retrieve a posted job, data will be displayed when the correct job name is entered.

Here you can preview the jobs on the network and the address of the ones who posted it. Also, Once you've done the transaction, you can see the history of the transactions with time and address by clicking on Show Transactions history button.

2.7 Post Experience

This is page where you can post your experience to our EKO-Pops Node.

Let's add some data and upload the experience

Here you can see the demo of how the transaction hash is generated for our transaction on the network. You can retrieve the transaction hash and check the status on our node or the status of the transaction will be shown at the end of the transaction as you'll see below.

For a successful transaction, you need to have your account address approved by the admin and then have some token energy. You can see the token being energy updated at the top right corner with the successful transaction. Once you've done the transaction, you can see the history of the transactions with time and status by clicking on Show Transactions history button. If any transaction fails, you'll have an option to resubmit that transaction without having to enter the credentials again. And if the transaction is successful, you'll see the the updated time and status in the transaction history table.

EKO Link

Address: 25c8...3d05

Amount: 0 PoPS Energy

Single User

Submit Credentials

Query Credentials

Submit Job Posting

Query Job Posting

Submit Experience

Query Experience

Bulk Upload

Logout

Employee Address:

Employee Address

Employee Experience:

Years

Experience in:

Skill: Java, MySQL, etc.

Submit experience

Show Transaction History

Copyright © 2018, EchoLink Project

EKO Link

Address: 25c8...3d05

Amount: 99.9993 PoPS Energy

Single User

Submit Credentials

Query Credentials

Submit Job Posting

Query Job Posting

Submit Experience

Query Experience

Bulk Upload

Logout

Employee Address:

0x25c891e8c4083b7a91daeb14e67ededec4f3d05

Employee Experience:

2

Experience in:

NodeJS|

Submit experience

Show Transaction History

Copyright © 2018, EchoLink Project

2.8 Query Experience

In this section of our application you can search for the addresses of people who match the experience and the job needs on our EKO-Pops Node. The image below shows the screenshot of the page.

Let's find the experience information which we posted in our previous section on *post experience*.

Here you can see the demo of how you can retrieve a posted experience, data will be displayed when the correct experience details are entered.

Here you can preview the addresses of required experience and the address of the ones who posted it. Also, Once you've done the transaction, you can see the history of the transactions with time and address by clicking on Show Transactions history button.

2.9 Bulk Upload Credentials

In this section of our application, we can upload the credentials as shown in our *upload credentials* demo in bulk.

As shown in the figure below, It displays two sections, one to upload a csv file and other to upload images.

In the first section, a .csv file < 50 entries containing fields as shown below can be uploaded. Please make sure that the image name field name must match with the images which you'll upload in the next section.

Issuer	Recipient	Status	Image
--------	-----------	--------	-------

eg:

TechRacers	EchoLink	Success	image.png
------------	----------	---------	-----------

In second section, images can be uploaded with .png | .jpg | .jpeg format and with file size < 500kb. The images name should match with the entries in csv file uploaded.

EKO Link

Address: 25c8...3d05

Amount: 99.9991 PoPS Energy

Single User

Submit Credentials

Query Credentials

Submit Job Posting

Query Job Posting

Submit Experience

Query Experience

Bulk Upload

Logout

Employee Experience:

2

Experience in:

NodeJS

Load experience

Show Transaction History

Copyright © 2018, EchoLink Project

EKO Link

Address: 25c8...3d05

Amount: 0 PoPS Energy

Single User

Submit Credentials

Query Credentials

Submit Job Posting

Query Job Posting

Submit Experience

Query Experience

Bulk Upload

Logout

Upload CSV file:

Choose file

No file chosen

*(Max number of rows 50)

Upload Images:

Choose Files

No file chosen

*(Max number of images 50 & size:500k/img)

Upload

Show Transaction History

Copyright © 2018, EchoLink Project

Single User ▶

Bulk Upload

Logout

Address: 25c8...3d05

Amount: 100 PoPS Energy

Upload CSV file:

Choose file certificates.csv

*(Max number of rows 50)

Upload Images:

Choose Files 5 files

*(Max number of Images 50 & size:500k(bimg))

Upload

Show Transaction History

Issuer	Recipient	Status	Preview
Anthony	Ajay	Graduated	
Estelle	Elaine	Graduated	
Barack	Bhanu	Stalled	

Copyright © 2018, EchoLink Project

As you can see above, once you select valid files the preview of the credentials will be displayed and you can choose to remove the credentials which you don't want to upload.

On clicking the Upload button in the bottom, a spinner will be displayed showing the progress and transactions will happen which will save images to IPFS and credential data to EKO-Pops Node. The demo below will show you how the transaction takes place.

For a successful transaction, you need to have your account address approved by the admin and then have some token energy. You can see the token being energy updated at the top right corner with the successful transaction. Once you've done the transaction, you can see the history of the transactions with time and status by clicking on Show Transactions history button. If any transaction fails, you'll have an option to resubmit that transaction without having to enter the credentials again. And if the transaction is successful, you'll see the the updated time and status in the transaction history table.

3.1 Steps to start the services

- Create a file `.env` to save the environment variables for the services
- Add and edit the following to the `.env`

```
BRIDGE_BLOCKCHAIN_HOST=  
BRIDGE_BLOCKCHAIN_PORT=  
BLOCKCHAIN_NODE_ID=  
BLOCKCHAIN_NODE_SECRET=  
MAINNET_BLOCKCHAIN_HOST=  
MAINNET_BLOCKCHAIN_PORT=  
EKO_CONTRACT_ADDRESS=
```

- Install `make` if not installed
- Run,

```
make set-up
```

This will initialize all the services and seed the database with the admin's account.

- To start all the services, run,

```
make dirty-up
```

You can access the services,

- [Token Bridge](#)
- [Reverse Token Bridge](#)
- [Eko Energy Transfer](#)

3.2 Steps to use the Eko Transfer service

- Enter the address to which the EKO Energy is to be transferred under the label `EKO To wallet address`
- Enter the amount under the label `EKO transfer amount`
- Enter your private key under the label `Your private key`
- Click on `Transfer` to perform the transfer transaction

This will transfer the provided EKO Energy to the requested address on the EKO network.

3.3 Steps to use the Token Bridge service

- For users
 - Sign up and login to the Token Bridge service
 - Transfer the EKO tokens to the address as shown under `Send your EKO Tokens here` section
 - Get the transaction hash of the transfer transaction and submit it to the section `Give us the transaction hash`.

The admin will now be notified with the transaction details and after verifying and processing it the user will receive the EKO Energy over the EKO network.

- For admin
 - Login to the Token Bridge service
 - Verify the transactions listed under *Submitted* section by clicking `Verify All Transaction` or verify them individually.
 - If the transaction is valid then, it will appear under the `Verified` section. Now, process the transactions by clicking *Process All Transaction* or process them individually.
 - Enter the private key to sign the transaction and perform the EKO network transaction.

If everything is valid the transaction will go through and the user will receive the EKO Energy over the EKO network.

3.4 Steps to use the Reverse Token Bridge service

- For users
 - Sign up and login to the Reverse Token Bridge service
 - Transfer the EKO Energy to the address as shown under `Send your EKO Energy here` section
 - Get the transaction hash of the transfer transaction and submit it to the section `Give us the transaction hash`.

The admin will now be notified with the transaction details and after verifying and processing it the user will receive the requested EKO tokens over the main net.

- For admin
 - Login to the Reverse Token Bridge service
 - Verify the transactions listed under *Submitted* section by clicking `Verify All Transaction` or verify them individually.

- If the transaction is valid then, it will appear under the `Verified` section. Now, process the transactions by clicking *Process All Transaction* or process them individually.
- Enter the private key to sign the transaction and perform the main net EKO token transfer transaction.

If everything is valid the transaction will go through and the user will receive the requested EKO token over the main net.

Authority Management For EKO Platform

Steps to start a network and implement validator set contracts to dynamically manage authorities:

4.1 Steps to set up the EKO network and dynamically manage network authorities:

1. Use the validator set contracts available at <https://github.com/parity-contracts/kovan-validator-set>
2. We just need these contracts:
 - OwnedSet.sol
 - BaseOwnedSet.sol
 - ValidatorSet.sol
 - Owned.sol
3. Setup the basic chain genesis saved under genesis.json with all required fields for Authority Round consensus engine

```
{
  "name": "DemoPoA",
  "engine": {
    "authorityRound": {
      "params": {
        "stepDuration": "1",
        "validators": {
          "list": []
        }
      }
    }
  },
  "params": {
    "gasLimitBoundDivisor": "0x400",
```

(continues on next page)

(continued from previous page)

[illegible]

4. Setup a node by creating a node config file saved under node.toml

```
[parity]
chain = "genesis.json"
base_path = "./eko_node_data"
[network]
port = 30300
[rpc]
cors = ["*"]
interface = "0.0.0.0"
port = 8545
apis = ["web3", "eth", "net", "personal", "parity", "eko_set", "traces", "rpc", "eko_
↳accounts"]
[ui]
interface = "0.0.0.0"
port = 8181
[websockets]
port = 8451
```

5. Start the node using `./eko --config node.toml --nat none`
6. Attach the geth to the exposed RPC port of the node using `geth attach http://localhost:8545`.
7. Create a new account using `personal.newAccount()` and an account password.

8. Create a file with the node passwords saved as node.pwds and add the password (assuming the password was “eko”) as a list

```
> eko
```

9. Update the node.toml as follows (assuming the generated address address is 0x00f3b949bb87ae90574c22f986c34207157b66b2)

```
[parity]
chain = "genesis.json"
base_path = "./eko_node_data"
[network]
port = 30300
[rpc]
cors = ["*"]
interface = "0.0.0.0"
port = 8545
apis = ["web3", "eth", "net", "personal", "parity", "eko_set", "traces", "rpc", "eko_
→accounts"]
[ui]
interface = "0.0.0.0"
port = 8181
[websockets]
port = 8451
[account]
password = ["node.pwds"]
[mining]
engine_signer = "0x00f3b949bb87ae90574c22f986c34207157b66b2"
reseal_on_txs = "none"
```

10. Restart the node using ./eko --config node.toml --nat none

11. Update the validator section of the genesis.json as follows

```
"validators": {
  "safeContract": "0x0000000000000000000000000000000000000000000000000000000000000005"
}
```

Such that the final genesis file becomes

```
{
  "name": "EKOPOA",
  "engine": {
    "authorityRound": {
      "params": {
        "gasLimitBoundDivisor": "0x400",
        "stepDuration": "1",
        "validators": {
          "safeContract": "0x0000000000000000000000000000000000000000000000000000000000000005"
        }
      }
    }
  },
  "params": {
    "gasLimitBoundDivisor": "0x400",
    "maximumExtraDataSize": "0x20",
    "minGasLimit": "0x1388",
    "networkID": "0x2323",
```

(continues on next page)

(continued from previous page)

[illegible]

The address specified in the `safeContract` address will be the deployed address of the validator set contract.

12. Refer to the validator set contracts at <https://github.com/parity-contracts/kovan-validator-set>

```
git clone https://github.com/parity-contracts/kovan-validator-set.git
cd kovan-validator-set
remixd -s contracts/ --remix-ide "https://remix.ethereum.org"
```

Go to <http://remix.ethereum.org/>

13. Open the localhost connection from the top left corner




```
event ValidatorAdded(address indexed validatorAddress, uint stake);
event ValidatorRemoved(address indexed validatorAddress, uint stake);
event CorruptValidatorRemoved(address indexed validatorAddress, uint stake);
```

- Update the addValidator function as follows

```
function addValidator(address _validator)
    external
    onlyOwner
    isNotValidator(_validator)
    payable
{
    require(msg.value == stakeAmount);

    status[_validator].isIn = true;
    status[_validator].index = pending.length;
    pending.push(_validator);
    validatorStake[_validator] = stakeAmount;

    triggerChange();

    emit ValidatorAdded(_validator, stakeAmount);
}
```

- Define a function removeCorruptValidator to add the functionality to remove a corrupt validator from the network and transfer the locked stake amount to the admin. The final function should be as follows,

```
function removeCorruptValidator(address _validator)
    external
    onlyOwner
    isValidator(_validator)
{
    // Remove validator from pending by moving the
    // last element to its slot
    require(validatorStake[_validator] > 0);

    uint index = status[_validator].index;
    uint _stakeAmount = validatorStake[_validator];

    pending[index] = pending[pending.length - 1];
    status[pending[index]].index = index;
    delete pending[pending.length - 1];
    pending.length--;

    msg.sender.transfer(_stakeAmount);
    validatorStake[_validator] = 0;

    // Reset address status
    delete status[_validator];

    triggerChange();

    emit CorruptValidatorRemoved(_validator, _stakeAmount);
}
```

- Update the function removeValidator to add the functionality to remove a corrupt validator from the network and transfer the locked stake amount to the admin. The final function should be as

follows,

```
function removeValidator(address _validator)
    external
    onlyOwner
    isValidator(_validator)
{
    require(validatorStake[_validator] > 0);

    // Remove validator from pending by moving the
    // last element to its slot
    uint index = status[_validator].index;
    uint _stakeAmount = validatorStake[_validator];

    pending[index] = pending[pending.length - 1];
    status[pending[index]].index = index;
    delete pending[pending.length - 1];
    pending.length--;

    msg.sender.transfer(_stakeAmount);
    validatorStake[_validator] = 0;

    // Reset address status
    delete status[_validator];

    triggerChange();

    emit ValidatorRemoved(_validator, _stakeAmount);
}
```

18. Select OwnedSet contract from the list of contracts to deploy.

19. In the arguments section, `_initial` would contain the list of initial validators. Here you need to place the array of addresses of your validator accounts. We will use `["0x00f3b949bb87ae90574c22f986c34207157b66b2"]` as we had assigned earlier to our 1st node config file. `_owner` should contain the address of the owner address (for this example we will be using `0x00f3b949bb87ae90574c22f986c34207157b66b2`) to which you would like to give authority to manage authorities in the network. Also, set the stake amount for all the new authorities in the network. For now we will be setting it to be equal to 200.

20. Copy the bytecode of contract along with the encoded values of input fields by clicking the briefcase button.

21. Update the account section for the `genesis.json` as follows,

```
"accounts": {
  . "0x0000000000000000000000000000000000000000000000000000000000000001": { "balance": "1", "builtin": {
  ↪ "name": "ecrecover", "pricing": { "linear": { "base": 3000, "word": 0 } } } },
    "0x0000000000000000000000000000000000000000000000000000000000000002": { "balance": "1", "builtin": {
  ↪ "name": "sha256", "pricing": { "linear": { "base": 60, "word": 12 } } } },
    "0x0000000000000000000000000000000000000000000000000000000000000003": { "balance": "1", "builtin": {
  ↪ "name": "ripemd160", "pricing": { "linear": { "base": 600, "word": 120 } } } },
    "0x0000000000000000000000000000000000000000000000000000000000000004": { "balance": "1", "builtin": {
  ↪ "name": "identity", "pricing": { "linear": { "base": 15, "word": 3 } } } },
    "<owner_address_holding_premined_ethers>": { "balance": "<provide_initial_
  ↪premined_ether_balance_here>" },
    "0x0000000000000000000000000000000000000000000000000000000000000005": { "balance": "<provide_initial_
  ↪balance_here>", "constructor": "<paste_byte_code_here>" }
}
```

Compile
Run
Settings
Analysis
Debugger
Support

Environment
JavaScript VM
VM (-)
i

Account
0xca3...a733c (100 ether)

Gas limit
3000000

Value
0
wei

OwnedSet
OwnedSet
BaseOwnedSet
Owned
ValidatorSet

Transactions recorded: 0


OwnedSet ▼ ⓘ

Deploy ^

_initial:

_owner:

_stakeAmount:

 **transact**

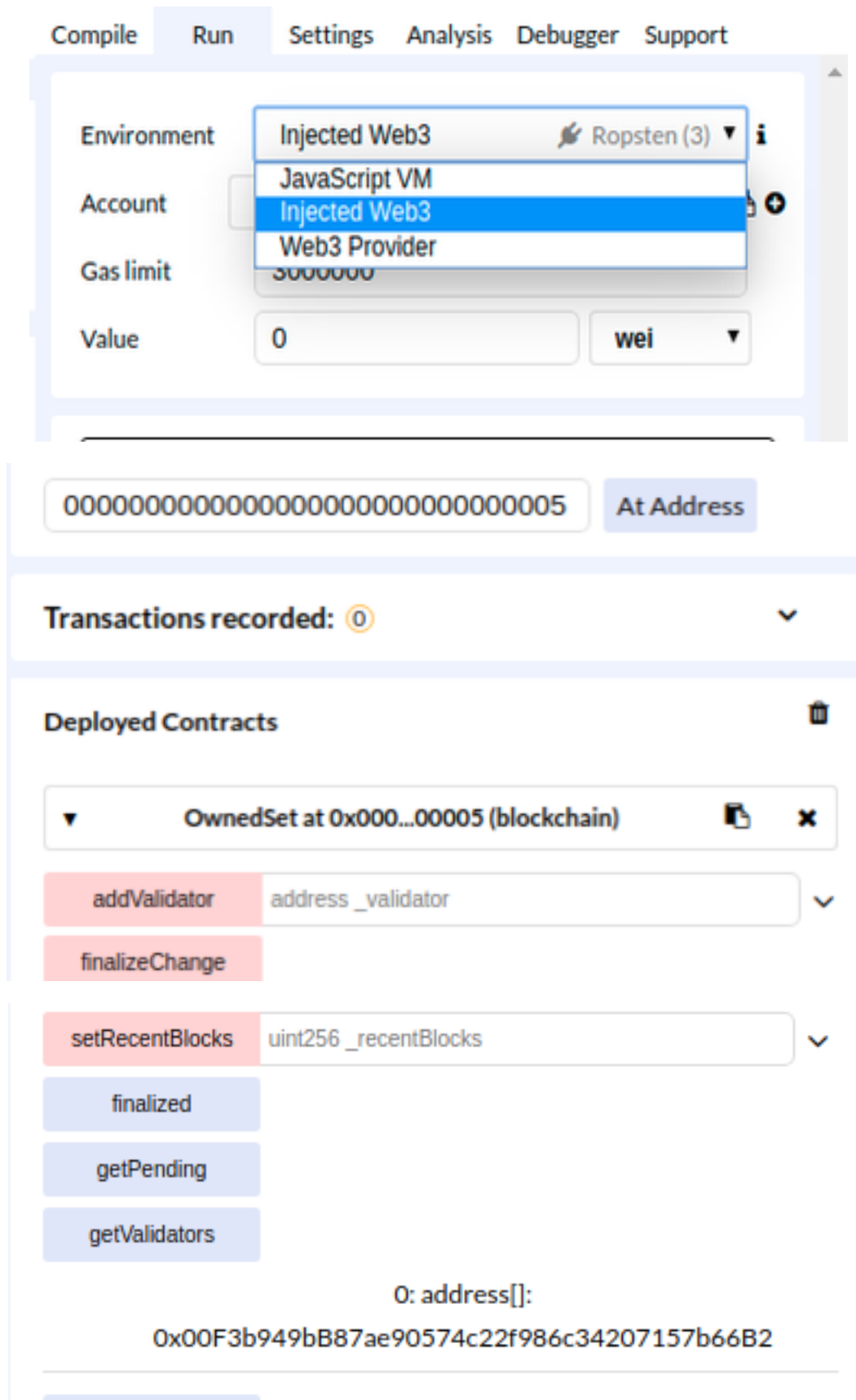
or

At Address

22. Restart the Eko nodes with the keys of the 0x00f3b949bb87ae90574c22f986c34207157b66b2 account.
23. Connect the metamask to the exposed RPC port of the nodes.
24. Import the account 0x00f3b949bb87ae90574c22f986c34207157b66b2 using its private key to the metamask accounts list.
25. Select Injected Web3 as the preferred environment in remix solidity browser.
26. Access and interact with the validator set contracts using At Address functionality of remix solidity browser, the contracts are predeployed at 0x00
27. Check the current validator by calling `getValidators` and the owner by calling `owner` constant functions
28. To add a new authority in the network, Copy the genesis file and start the second node with the node config saved under `node.toml` (on 2nd nodes system)

```
[parity]
chain = "../genesis.json"
base_path = "./eko_node_data1"
[network]
port = 30301
[rpc]
cors = ["*"]
interface = "0.0.0.0"
port = 8541
apis = ["web3", "eth", "net", "personal", "parity", "eko_set", "traces", "rpc", "eko_
↪accounts"]
[ui]
```

(continues on next page)



(continued from previous page)

```
interface = "0.0.0.0"
port = 8181
[websockets]
port = 8451
```

29. Connect the nodes, Here we will simply use curl. Obtain 1st node's enode:

```
curl --data '{"jsonrpc":"2.0","method":"parity_enode","params":[],"id":0}' -H
"Content-Type: application/json" -X POST localhost:8545
```

30. Add the result to node 1 (replace enode://RESULT in the command):

```
curl --data '{"jsonrpc":"2.0","method":"parity_addReservedPeer",
"params":["enode://RESULT"],"id":0}' -H "Content-Type: application/json" -X
POST localhost:8541
```

Now the nodes should indicate 1/25 peers in the console, which means they are connected to each other.

31. Geth attach to the node's RPC exposed port using `geth attach http://localhost:8541`

32. Generate a new account using `personal.newAccount()`. Choose a password for the account.

33. Update the node config file for the second node by assigning the generated account value (assuming the generated account is `0x8c7cfb7f40b7a6c4d34c7619c6075d0402112811`) to the `engine_signer` such that the node config file looks like,

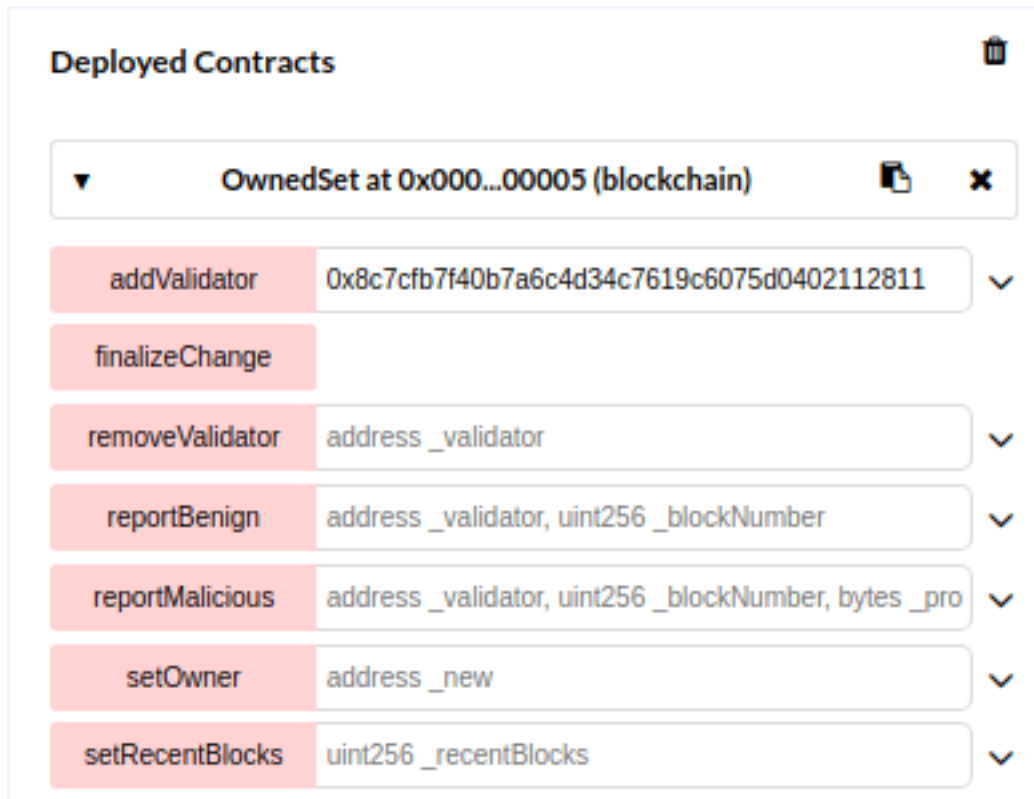
```
[parity]
chain = "../genesis.json"
base_path = "./eko_node_data1"
[network]
port = 30301
[rpc]
cors = ["*"]
interface = "0.0.0.0"
port = 8541
apis = ["web3", "eth", "net", "personal", "parity", "eko_set", "traces", "rpc", "eko_
↪accounts"]
[ui]
interface = "0.0.0.0"
port = 8181
[websockets]
port = 8451
[account]
password = ["node.pwds"]
[mining]
engine_signer = "0x8c7cfb7f40b7a6c4d34c7619c6075d0402112811"
reseal_on_txs = "none"
```

34. Restart the second node

35. Sign in with the owner account using the metamask and select the Injected Web3 from the environment dropdown in remix solidity browser.

36. Now, access the OwnedSet.sol contract again as before using the predeployed contract address and perform the transaction `addValidator` with address parameter `0x8c7cfb7f40b7a6c4d34c7619c6075d0402112811` to add the second node's owner address as a new authority.

Note: the stake amount as `msg.value` needs to be supplied with this transaction



37. The node logs should look like,

```
1/25 peers 1 MiB chain 86 KiB db 0 bytes queue 10 KiB sync RPC: 0 conn, 2 req/s, 57 µs
Signal for transition within contract. New list: [00f3b949bb87ae90574c22f986c34207157b66b2, 8c7cfb7f40b7a6c4d34c7619c6075d0402112811]
Applying validator set change signalled at block 42
Transaction mined (hash 5980160ee23df24798af1830aa9e27cd6bf1936b6985d338359a9ca7a06d8626)
Imported #42 e998_de6d (1 txs, 0.10 Mgas, 1.31 ms, 0.70 KiB)
```

38. We can check the stakes for the validator using the constant function `validatorStake` as,

39. Now, if we call `getValidator` we should get

This gives a confirmation that the new authority has been added to the network.

40. To remove an authority the owner should perform the transaction `removeValidator` with address parameter `0x8c7cfb7f40b7a6c4d34c7619c6075d0402112811`. The stakes will be transferred to the authority's address from the contract and the validator will be removed from the network.

The node logs should look like,

Now, if we call `getValidator` we should get

This gives a confirmation that the mentioned authority has been removed from the network.

When we check the stake balance now, we should get

41. To remove a corrupt authority the owner should perform the transaction `removeCorruptValidator` with the address parameter `0x8c7cfb7f40b7a6c4d34c7619c6075d0402112811`. In this case, the stake will not be transferred to the authority's address but these will be transferred to the owner's address and the validator will be removed from the network.

stakeAmount

O: uint256: 200

systemAddress

validatorStake 0x8c7cfb7f40b7a6c4d34c7619c6075d0402112811

O: uint256: 200

getPending

getValidators

O: address[]:

0x00F3b949bB87ae90574c22f986c34207157b66B2,0x8c7cfB7
F40B7A6C4d34C7619C6075d0402112811

Deployed Contracts



OwnedSet at 0x000...00005 (blockchain)



addValidator address_validator

finalizeChange

removeValidator 0x8c7cfb7f40b7a6c4d34c7619c6075d0402112811

reportBenign address_validator, uint256_blockNumber

reportMalicious address_validator, uint256_blockNumber, bytes_pro

setOwner address_new

setRecentBlocks uint256_recentBlocks

```
Imported #67 96c0...ef7f (1 txs, 0.02 Mgas, 0.55 ms, 0.67 KiB)
1/25 peers 41 KiB chain 111 KiB db 0 bytes queue 20 KiB sync RPC: 0 conn, 0 req/s, 0 µs
1/25 peers 41 KiB chain 111 KiB db 0 bytes queue 20 KiB sync RPC: 0 conn, 0 req/s, 0 µs
Imported #68 2e66...2d76 (1 txs, 0.02 Mgas, 0.55 ms, 0.67 KiB)
Signal for transition within contract. New list: [00f3b949bb87ae90574c22f986c34207157b66b2]
Imported #69 a5a6...9449 (1 txs, 0.02 Mgas, 0.46 ms, 0.67 KiB) + another 1 block(s) containing 1 tx(s)
```

setRecentBlocks

uint256 _recentBlocks

▼

finalized

getPending

getValidators

O: address[]:

0x00F3b949bB87ae90574c22f986c34207157b66B2

stakeAmount

O: uint256: 200

systemAddress

validatorStake

0xA84e4a475082bEbab053a53866438B2a30c50426

▼

O: uint256: 0

4.2 Using Docker to setup the nodes in the network

1. Clone the EKO platform repository
2. Run

```
make set-up
```

This will install all the dependencies and set up build the docker image for the EKO node

3. Copy the genesis configurations for the network to the `blockchain/configurations/genesis.json`
4. Copy the key in file format to the `blockchain/configurations/key.json`. If the node is set up as the miner node then this will be used.
5. Copy the password used to create the `key.json` to the `blockchain/configurations/node.pwds`.
6. Different modes:

- Start a new network

1. Run

```
make initialize-blockchain
```

2. Type 1 and enter to select Start a new network from the options provided.

3. Run

```
make dirty-up
```

This will start the node with a new network configurations.

- Join an existing network as a miner node

1. Run

```
make initialize-blockchain
```

2. Type 2 and enter to select Join an existing network as a miner node from the options provided.

3. Enter the valid enode value of the node you would like to connect to from the existing network

4. Run

```
make dirty-up
```

This will start the node as a miner node and sync with the existing nodes in the network. If the validator address has been given permission to become a validator in the network the node will start mining the new blocks, else it will wait for the admin to grant permission using the Validator set contract methods.

- Join an existing network as a viewer node

1. Run

```
make initialize-blockchain
```

2. Type 3 and enter to select Join an existing network as a viewer node from the options provided.

3. Enter the valid enode value of the node you would like to connect to from the existing network

4. Run

```
make dirty-up
```

This will start the node as a viewer node and sync with the existing nodes in the network.

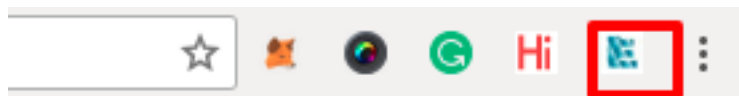
7. Now, follow the steps as mentioned from points 31 to 41 from the previous section.

EkoLink Extension takes the information of a webpage. It provides the image, ssl info and content of full web page and save these information to ekolink backend. You can find extension on chrome store.

5.1 How to use extension

To use ekolink extension, add extension to your chrome browser and follow the instruction below.

Navigate on a web page and activate the extension by clicking on extension icon.



Initial page - Get user info (user name) before capturing credential.

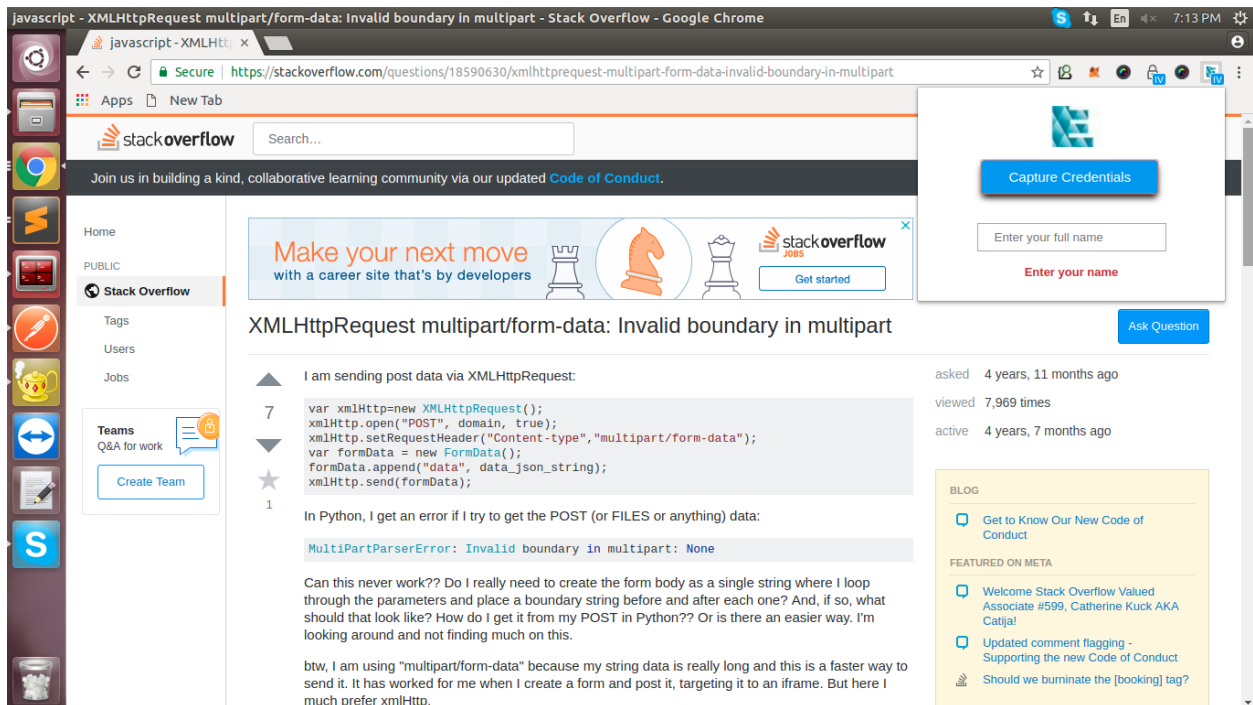
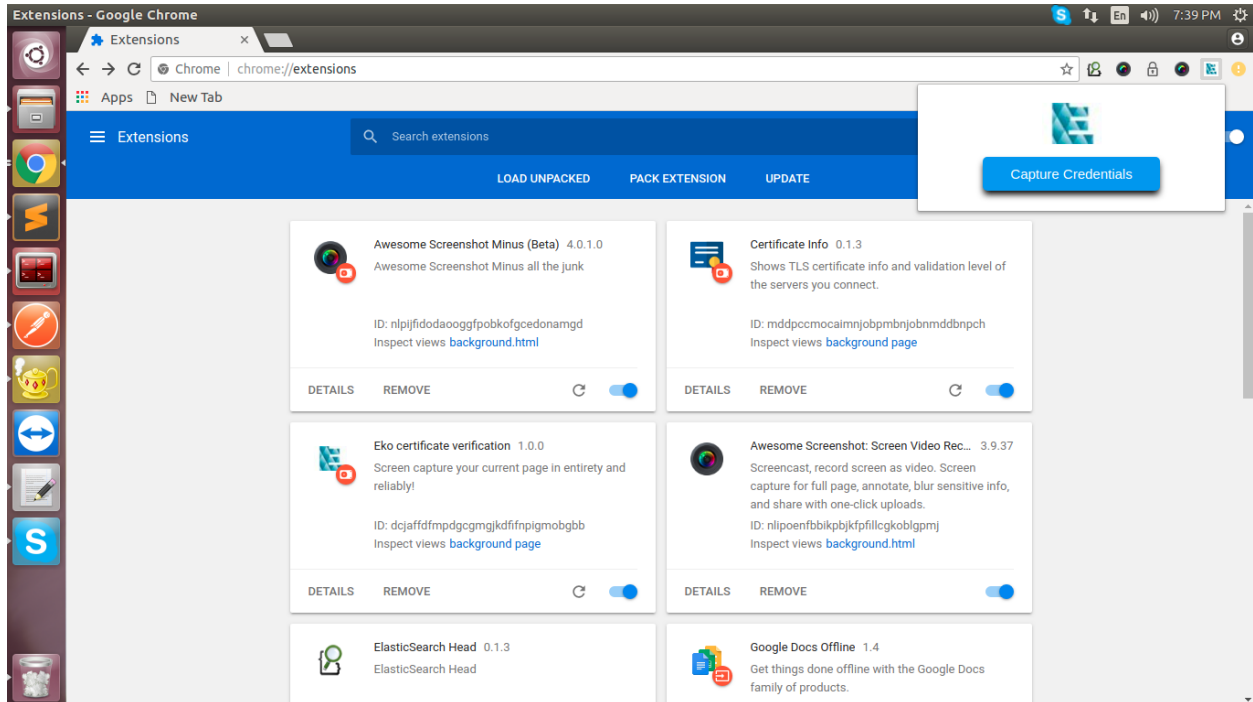
If you click on capture credential button without input your name, it will give the error.

Input the name and then click on **Capture Credential** button.

If user is on other then http or https web page he will get error.

After clicking on submit button will be disable and show status saving.

You will get Success message, after your credential successfully get submitted.



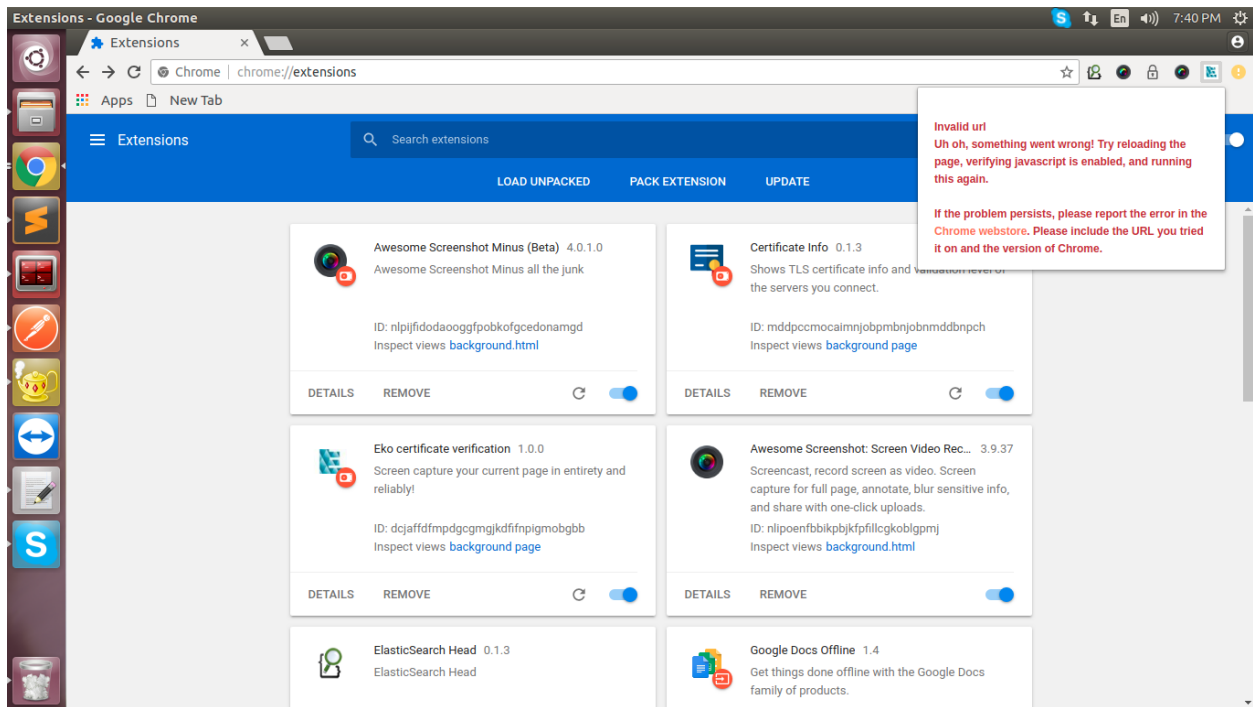


Fig. 1: On https/http web page

