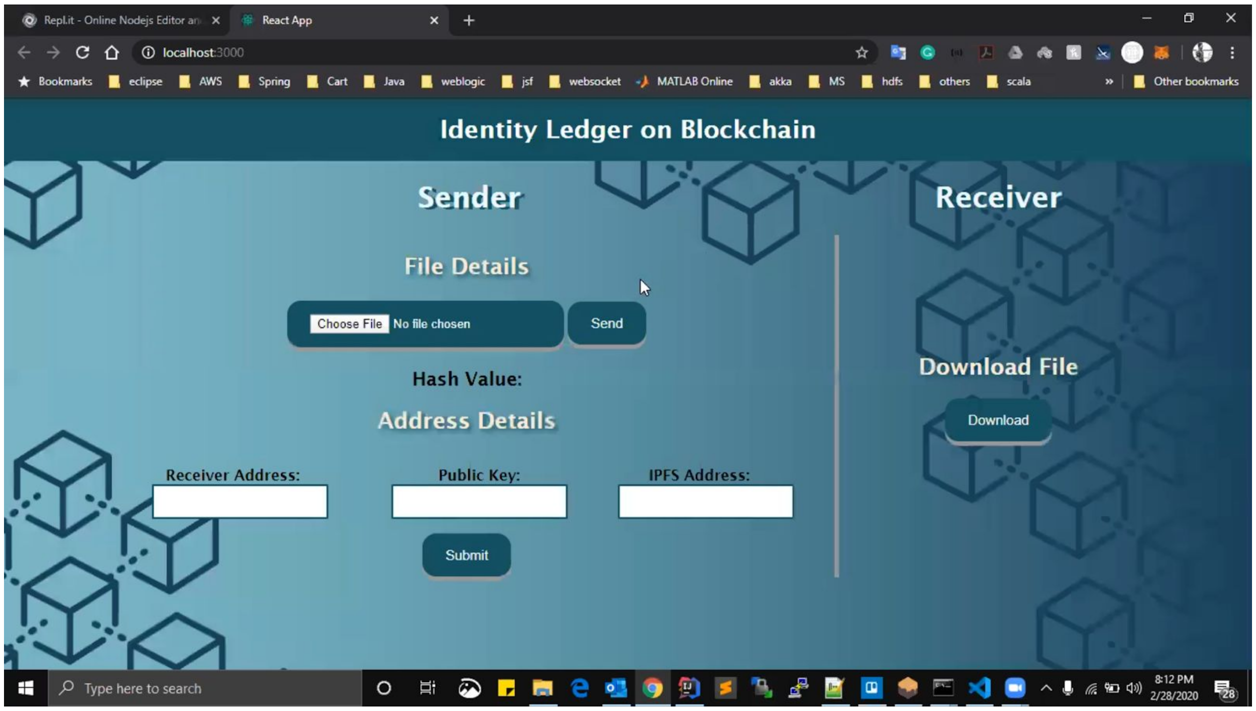
This document walks through the main steps of the video demonstration of the project -



Receiver Address: Ganache account address of the intended recipient.

Public Key: It is the public key of the recipient obtained by using the recipient's private key. We have used 'publicKeyByPrivateKey()' function from 'eth-crypto' library in node.js to fetch the recipient's public key. This function takes the recipient's private key as a parameter to generate the public key.

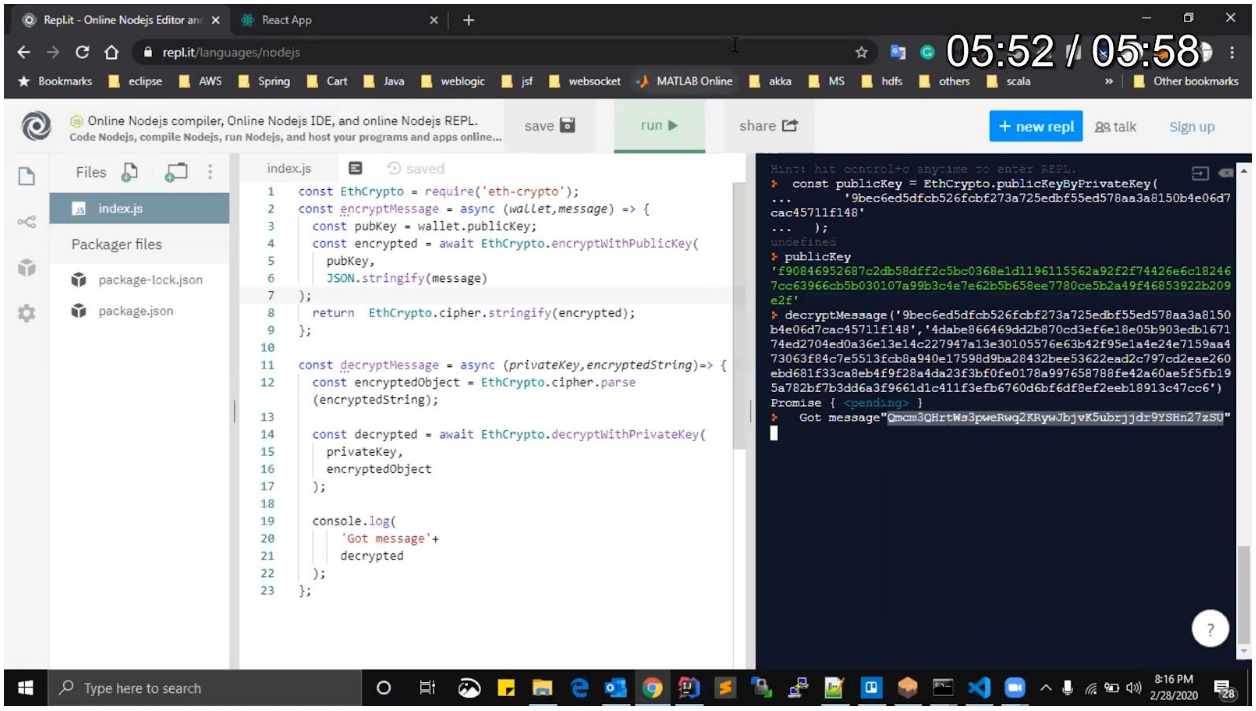
```
const publicKey = EthCrypto.publicKeyByPrivateKey(
        '9bec6ed5dfcb526fcbf273a725edbf55ed578aa3a8150b4e06d7cac45711f148'
    );
```

IPFS address: We used 'ipfs-api' module in node.js to store the file in IPFS and get its hash code.

When we click on the 'Submit' button, the IPFS address is encrypted with the recipient's public key. It is done as follows, using the 'eth-crypto' library in node.js.

```
const encryptMessage = async (pubKey,message) => {
const encrypted = await EthCrypto.encryptWithPublicKey(
    pubKey, // by encryping with bobs publicKey, only bob can decrypt the payload with his privateKey
JSON.stringify(message) // we have to stringify the payload before we can encrypt it
);
  return  EthCrypto.cipher.stringify(encrypted);
};
```

Here, after the recipient clicks the ‘Download’ button, he gets the encrypted IPFS hash of the intended file.



```
const EthCrypto = require('eth-crypto');
const encryptMessage = async (wallet,message) => {
  const pubKey = wallet.publicKey;
  const encrypted = await EthCrypto.encryptWithPublicKey(
    pubKey,
    JSON.stringify(message)
);
  return  EthCrypto.cipher.stringify(encrypted);
};

const decryptMessage = async (privateKey,encryptedString)=> {
  const encryptedObject = EthCrypto.cipher.parse
  (encryptedString);

  const decrypted = await EthCrypto.decryptWithPrivateKey(
    privateKey,
    encryptedObject
  );

  console.log(
      'Got message'+
      decrypted
  );
};
```

```
Hint: hit control+c anytime to enter REPL.
> const publicKey = EthCrypto.publicKeyByPrivateKey(
...       '9bec6ed5dfcb526fcbf273a725edbf55ed578aa3a8150b4e06d7
cac45711f148'
...   );
undefined
> publicKey
'f90846952687c2db58dff2c5bc0368e1d1196115562a92f2f74426e6c18246
7cc63966cb5b030107a99b3c4e7e62b5b658ee7780ce5b2a49f46853922b209
e2f'
> decryptMessage('9bec6ed5dfcb526fcbf273a725edbf55ed578aa3a8150
b4e06d7cac45711f148','4dabe866469dd2b870cd3ef6e18e05b903edb1671
74ed2704ed0a36e13e14c227947a13e30105576e63b42f95e1a4e24e7159aa4
73063f84c7e5513fcb8a940e17598d9ba28432bee53622ead2c797cd2eae260
ebd681f33ca8eb4f9f28a4da23f3bf0fe0178a997658788fe42a60ae5f5fb19
5a782bf7b3dd6a3f9661d1c411f3efb6760d6bf6df8ef2eeb18913c47cc6')
Promise { <pending> }
>   Got message"Qmcm3QHrtWs3pweRwq2KRywJbjvK5ubrjjdr9YSHn27zSU"
```

The following code is used to decrypt and get the original IPFS hash using the private key. This hash will be used to get the file from IPFS.

```
const EthCrypto = require('eth-crypto');

const encryptMessage = async (publicKey,message) => {
  const encrypted = await EthCrypto.encryptWithPublicKey(
    pubKey, // by encryping with bobs publicKey, only bob can decrypt the
payload with his privateKey
    JSON.stringify(message) // we have to stringify the payload before we can
encrypt it
);
  return  EthCrypto.cipher.stringify(encrypted);
};

  const decryptMessage = async (privateKey,encryptedString)=> {
  const encryptedObject = EthCrypto.cipher.parse(encryptedString);

  const decrypted = await EthCrypto.decryptWithPrivateKey(
    privateKey,
    encryptedObject
  );

  console.log(
      'Got message'+
      decrypted
  );
};
```