CPR E 550
Project 3
Eranda Sooriyarachchi

System Design

Objective

Designing a Flight Insurance System on a Ethereum Blockchain Network.

Overview

The system is composed of 3 main parts:

1. Ethereum Network
2. Client (Front end)
3. Cloud fxn

Ethereum Network

This is the network where the smart contract of Ethereum is deployed. The Sepolia Test Network was chosen for deployment as that is what is used by many researchers in the field. The constructor function of the "Contract" is called when the contract is deployed. In the contract, the manager who will indemnify clients in the event of policy violations should be specified during deployment. When the deployment is complete, it delivers two key elements, namely, ABI and Address, that are needed for the front end and cloud functions to connect to our Ethereum Network.

ABI: ABI instructs the function's caller to encrypt the necessary data, such as function signatures and variable declarations, in a manner that the EVM can comprehend before invoking the function in JSON.

Address: This is the address of contract deployment.

Client

The client-side application, which will be written in React, is known as the front-end. To approve or reject transactions and keep the account in sync with the client, this application will communicate with MetaMask (a Chrome extension utility). We can get a Metamask wallet instance using the web3 package that is offered by npm. Therefore, we will be able to make transactions and get data such as account details and Ethereum balance. We also need the deployed contract's ABI and address in addition to this.

Cloud Function

The cloud function is a function that can do scheduled activities while using very little resources. This cloud function will retrieve the weather information (via an API or a text file)

and the client's destination city and dates. It will retrieve the relevant weather information and determine whether it is accurate. It will send the indemnification amount from the manager's account to the client's address if the weather is bad. We will utilize the "node-schedule" package since we want to carry out this work automatically without any human involvement. We will be able to execute the verification at regular intervals thanks to this.

The Contract

Simply said, a Smart Contract is a piece of code that executes on the Ethereum Network. It is a set of functions and state-related data that are stored at a particular address on the Ethereum blockchain. Components of Ethereum contract are described below.

Storage of Contract Data

The Mapping : The struct of the PolicyPurchased class, which contains all the necessary customer information such name, flight name, destination, departure date, etc., is used to establish the mapping between payable addresses of customers. This will make it easier to immediately retrieve the data when a consumer asks it.

The Address Lookup: It is an array of client addresses. It maintains a list of its active clients.

The Manager : This address component has the account information for the contract manager. When a contract is deployed in the constructor function, it is set.

The Customer Array : It is an array of the class PolicyPurchased. When the manager retrieves all the active policies, the array used to hold the whole data is returned.

Functions of Contract

Constructor : manager value is set during deployment by the constructor.

View_available_policy : With all the policies offered, no modifiers are necessary because it produces an array of strings.

Check Policy Status : This function determines if a customer address is now the owner of an insurance. This function's purpose is to prohibit users from purchasing numerous policies because doing so would cause the value already existent in the mapping to be replaced.

PurchasePolicy: This function enables customers to purchase insurance. Because transactions take place here, it's a payment function. It needs the minAmount modifier, which verifies if the user sent Ethereum with the request.
View_all_purchased_policy : The manager uses this function to find all the clients who have policies. Restricted modifier is needed since only the contract manager should manage it.

PayIndemnity : This method uses the customer's whole address list to send the indemnification to the specified addresses. Because only the manager may use this function, the modifier restricted is needed.

Modifiers implemented in Contract

Restricted: Checks if user who generated the request is verified as a manager or not. Users like clients who frequently use functions like view_all_policy and pay_Inmdenity are prevented from doing so.

minAmount: This modification verifies that the user sent at least 0.001 ether when purchasing the insurance.

max_one_policy: If this modifier is not present, the mapping will only allow one policy per user account.

The Client side Front-End

To access the "Metamask" instance, the client utilizes the web3 npm package. It supplies the web3 with the ABI and network address created during code deployment. This grants access to all the contract's methods.

Following fxns can access the contacts:

For call / view fxn:
instance.methods.methodname({parameters : params}).call({ from : account Address })

For payable fxn:
instance.methods.methodName({ parameters : params }).send({ from : account Address, value : amount_of_ether_ coin_ in_ wei })

The Cloud Function

The Contract can communicate with outside data thanks to the cloud function. For the manager, it implements the "verify" function.
The verify function is implemented as follows:
Retrieves all the data about policyholders from the deployed contract. Filter every city and flight date from the data retrieved from the previous section. Use the filtered data to contact the weather api. Remove addresses with unfavorable weather. Complete the payable method call (paylndeminty) in the contract and submit all the addresses that are required for the payment of indemnity as parameters. Include in the method call the minimal quantity of ether needed to pay all the clients.
We utilize the node-schedule package to make the aforementioned function execute automatically without human involvement. It enables the aforementioned action to take place as scheduled.

Performance

Fxn	**Time taken(s)**
View_policies	1.421
View_all_policy_holders	0.078
View _purchased_policy	0.125
Purchase_policy	22.807
Pay_Indemnity	7.510