Problem 3: Blockchain-enabled Flight Travel Insurance System

Due: 11:59PM, Wednesday, April 30, 2025 (same for both on- and off-campus students)

I. Overview

This course project is intended to familiarize you with the tools and with the process of building and deploying an Ethereum distributed application. The objective of this machine problem is to use the Ethereum platform to build a Blockchain-enabled Flight Travel Insurance System where a passenger can purchase flight insurance policies and will receive indemnity automatically when certain conditions are met.

Flight insurance is a coverage option that is most often available as an add-on when the passenger booking an airline ticket. Every flight insurance policy outlines what may be covered regarding unexpected events related to air travel and the maximum value of the coverage. In general, this type of policy covers flight delay, trip cancellations or trip interruption, and lost, damaged, or stolen luggage. In the case that covered accidents happened, the policyholder would be required to file a claim with the insurance provider for compensation. The flight insurance provider will then verify the claim. If the situation is true, it will pay indemnity in accordance with the provisions of the insurance policy.
The idea for this project is from AXA and Etherisc.

II. Project Requirements

This course project consists of three phases: Phase I project aims to develop a smart contract that can purchase flight insurance policies and file claims; Phase II project aims to develop a smart contract extended from Phase I that can verify claims using the provided weather data and pay indemnities when passengers lose; Phase III (extension) project aims to develop a smart contract that can automatically verify claims and pay compensation in the event of passenger losses through weather data extracted from the network.

A. Basic Requirements

For every student, finish the Phase I and phase II project.
Note: We expect each student will design and implement this programming assignment individually. Please schedule a time with the TA to show the project demo between May 1, 2025 to May 11, 2025.

B. Extra Credit Opportunities

In addition to the above basic requirements, you can receive additional credits (**up to 10 points, in addition to the 15 points for the basic requirements**) by implementing the extension (i.e., Phase III). We encourage students to make any meaningful attempts in this phase. Even if the requirements cannot be fully achieved, we will give partial points based on the completion. Detailed requirements for this project are outlined below.

III. Phase I - A Simple Smart Contract

The first phase of the project is intended to familiarize you with the tools and with the process of building and deploying a simple distributed application on Ethereum. For this phase, you are to develop a smart contract, where a passenger can view or purchase the flight insurance policy, and the insurance provider can view all purchased policies. The system must meet the following general requirements:

- There are two roles in the system: Passenger and Insurance Provider. They are in the same Ethereum network and can be identified by Ethereum addresses.

- The passenger role should provide the functionality specified by the following use case scenarios (illustrated in Figure 1):
  - **view_available_policy**: The passenger can query the available policy by invoking this function. The output of this function can be a string, which contains the premium, indemnity, and coverage issues. For the system simplicity, we use the fixed policy premium and indemnity, which are 0.01 and 0.02 Ether, respectively. Coverage issues include extreme weather such as hail and floods. This means that if the flight's **departure city** has hail or flood on the day of the flight, which prevents the flight from taking off normally, then passengers who purchase flight insurance can receive an indemnity of 0.02 Ether.

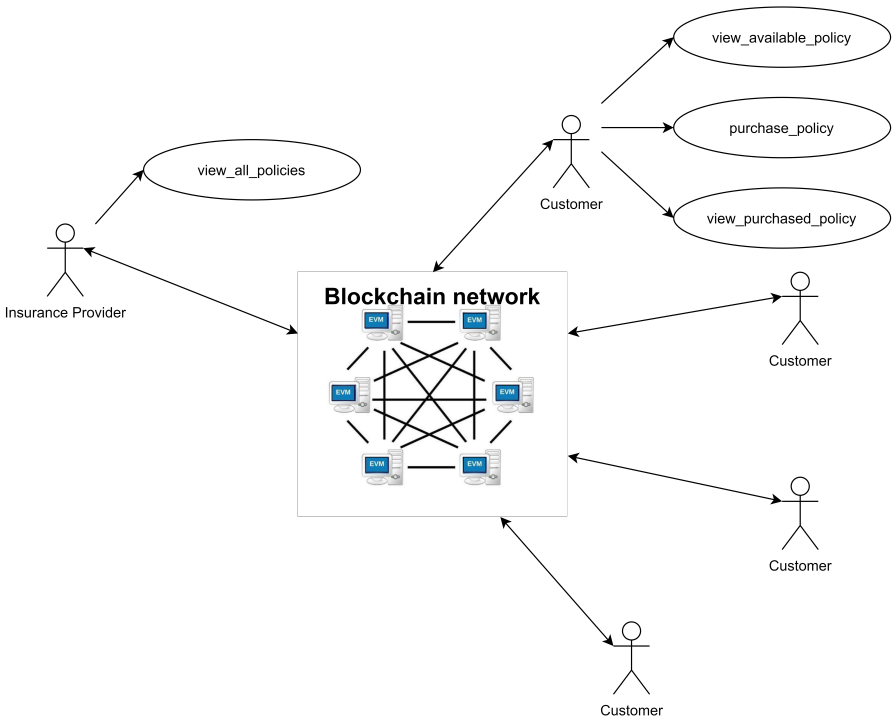

Fig. 1. Case Scenarios for passenger and Insurance Provider's Functionality for Phase I

- **purchase_policy**: After viewing the available policy, the passenger can purchase an insurance policy for their flight by invoking this function. The flight insurance policy should contain the following information: passenger's name, passenger's Ethereum address, flight number, flight date, departure city, destination city, and **policy status**. The policy status is a string with two possible values: "purchased" and "claimed". When the passenger invokes **purchase_policy** function, the policy status should be set to "purchased". This information can be provided as arguments when the passenger calls **purchase_policy** function. Simultaneously, the premium should also be sent from the passenger's account to the insurance provider's account when this function is invoked.
  In summary, this function does two things: First, create an insurance policy for the passenger; Second, pay the premium using the passenger's balance. Again, we assume that the premium is fixed (0.01 Ether), and the insurance provider's address is well known to all passengers.
  Tips: Consider using the "struct" to define the insurance policy's properties.

- **view_purchased_policy**: After purchasing the insurance policy, the passenger can invoke **view_purchased_policy** function to check their policy information. The output of this function should include all properties of an insurance policy, including passenger name, flight number, flight date, departure city, destination city, and policy status.
  For the system's simplicity, we assume that each passenger only purchases one ticket and can only purchase one insurance policy for that ticket. So, the system may use the passenger's name or address as the key to identify the passenger's policy and return it to the passenger. You can definitely use other approaches to help passengers to retrieve their insurance policies. We encourage you to develop your ideas, draw them in design documents, and implement them in smart contracts.

- The insurance provider role should provide the functionality specified by the following use case scenarios (illustrated in Figure 1):
  - **view_all_policies**: For the insurance provider, by calling this function, he should know all the purchased policies. For each policy, the output information should contain all its properties.
    Tips: For the storage patterns of the Solidity, we recommend you to read this page or to the next URL: https://ethereum.stackexchange.com/questions/13167/are-there-well-solved-and-simple-storage-patterns-for-solidity.
    Tips: The view_all function should be invoked only by the Insurance Provider. To implement this requirement, consider using the "function modifiers".

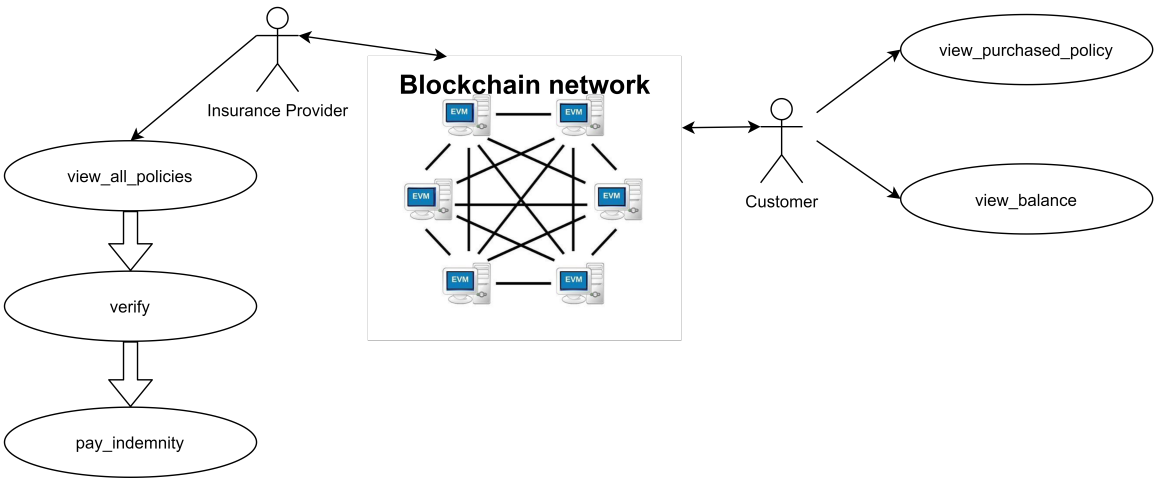

Fig. 2. Case Scenarios for passenger and Insurance Provider's Functionality for Phase II

IV. Phase II - A Flight Insurance System

The purpose of this phase is to extend our working knowledge of the Ethereum platform by implementing a flight insurance system that can use provided weather data to verify the loss of passengers and pay them indemnities.

The workflow for Phase II is described as Figure 2. First, the insurance provider can view or download the insurance policies purchased by all passengers. Next, it examines these policies and compares them with the provided weather data. If it is found that a passenger's flight is affected by the weather and cannot take off normally, the insurance provider needs to do two things: first, pay the indemnity (0.02 Ether) to the passenger; second, modify the status of the passenger's insurance policy to "claimed". Passengers can check their policy status by invoking the **view_purchased_policy** function implemented in Phase I and can verify the indemnity payout by invoking the **view_balance** function developed in this phase.
General requirements, beyond those of Phase I, are shown as follows:

- The insurance provider role should provide the functionality specified by the following use case scenarios:
  - **verify**: This function is to examine whether the passenger's flight is delayed using the provided weather data and the policy information on the Ethereum network. If extreme weather (hail, flood) occurs in the passenger departure city on that day, the flight will be deemed unable to be performed normally. We require students to implement the **verify** function to finish this process. In other words, this function cannot be completed by the user manually inputting weather conditions into the blockchain.

    Tips: Because the smart contract cannot process the weather data, we need to find a way to implement the **verify** function. You may refer to the "deploy_web3.js" program and the "access_web3.js" program provided on the Canvas and learn how to invoke a smart contract function using JavaScript. You can write a JavaScript program to finish the following steps: First, read the weather data from the provided text file; Second, read the policy data from the Ethereum network; Finally, using this two information to decide which policy should be paid. The above steps are just a reference; we also encourage students to try other implementation methods. If you have other ways to achieve this function, please clarify it in the design document and implement it through code.

  - **pay_indemnity**: This function should be called when the insurance provider has verified the passenger's flight delay. This function will send indemnity (0.02 Ether) to the passenger's account and return a message to indicate whether the payout was sent successfully. The returned message can be a Boolean value or a string that indicates whether the payment is sent.

    Tips: The **pay_indemnity** function should be implemented in the smart contract, and can be invoked after the **verify** function returns. You can extend the above JavaScript program to make the payment happen if the payout condition meets.

- The passenger role should provide the functionality specified by the following use case scenarios:
  - **view_balance**: This function is in the smart contract and should return the caller's current total assets (Ether). If the passenger's flight delay is verified by the insurance provider, the "policy status" of the insurance policy should be "claimed," and the indemnity should also be sent to the passenger. After the passenger calls **view_purchased_policy** function developed in Phase I and finds that the status has changed, he can verify the payout by calling this function (the total asset should increase 0.02 Ether). You can also add this function as a functional item to the **view_purchased_policy** function so that passengers can more easily observe their insurance payment status and total assets.
- We require students to create at least one flight insurance policy that meets the conditions of compensation so that the functions implemented in Phase II can be tested normally during the demo.

V. Phase III - A Fully Automated Flight Insurance System

The purpose of this phase is to master our working knowledge of the Ethereum platform by implementing an automated flight insurance system that can extract actual weather data to verify the loss of passengers and automatically pay them indemnities.

In Phase II, the insurance provider can use provided weather data to process the verify and payout steps, which still requires external data to drive the system. In Phase III, we encourage you to implement a fully automated flight insurance system that can meet the following requirements:

- The passenger can purchase the flight insurance policy, as defined in Phase I.
- The system can track each insurance contract and use the public weather API to capture local weather conditions according to their departure place and time.
  Tips: For the public weather API, we recommend you to read this page or to the next URL: https://www.weather.gov/documentation/services-web-api. Briefly speaking, such APIs allow developers access to critical forecasts, alerts, and observations, along with other weather data. They are designed with a cache-friendly approach that expires content based upon the information life cycle. You can search online and learn how to use JavaScript program to call these APIs, and how to handle the returned values.
- If some passengers' flights cannot take off due to extreme weather (such as hurricanes, rainstorms, etc.), the system will automatically determine that the policy is in effect and send compensation to the passengers according to the policy. The process should be performed entirely by the program, without human intervention.
- The passenger can verify the policy status and payment by querying the Ethereum blockchain.
- The detailed requirements of the insurance policy are the same as those explained in Phase I, including the premium, indemnity, and other properties.

We encourage students to make any meaningful attempts in this phase. Even if the requirements cannot be fully achieved, we will give partial points based on the completion.

VI. What to Hand In

A. Design

Before you start hacking away, plot down a design document. The result should be a system-level design document, which you hand in along with the source code. Please do not get carried away with it, but make sure it convinces the reader that you know how to attack the problem. The source code and reports should be submitted through Canvas.

Please schedule a time with TA Md Shafiqul Islam (shafiqul@iastate.edu)to show a virtual Zoom project demo between May 1, 2025, and May 11, 2025. Note that the demo requirement is both for on-campus and off-campus students. For students who do not show a demo, the project will be graded by the TA manually on Canvas.

B. Measurements

In order to compare the efficiency of your implementation, you will perform some measurements. I will leave it to you to decide what measurements should be done.