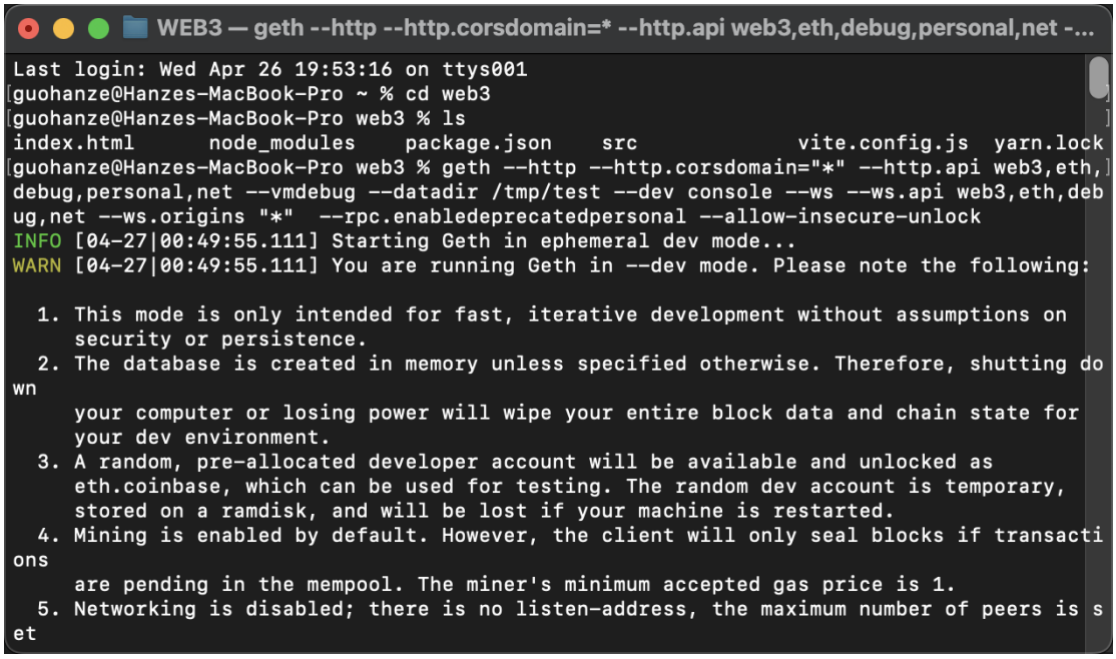
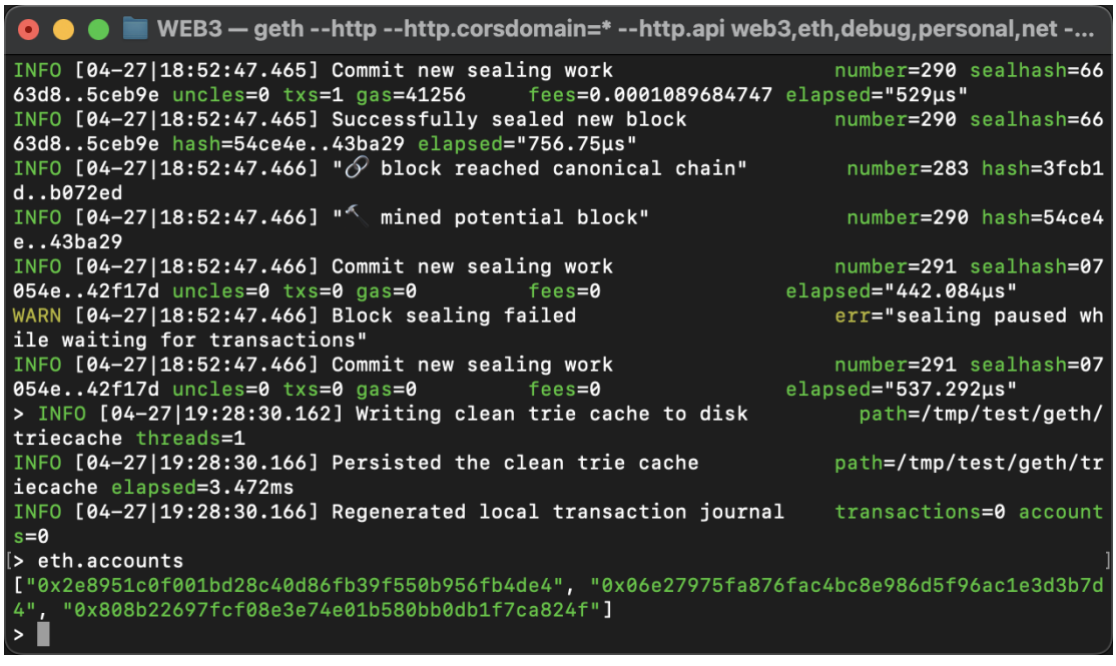
I rewrited ‘main.jsx’ file based on the code in PPTX, implemented the whole process and UI part, it can be started by the following steps.

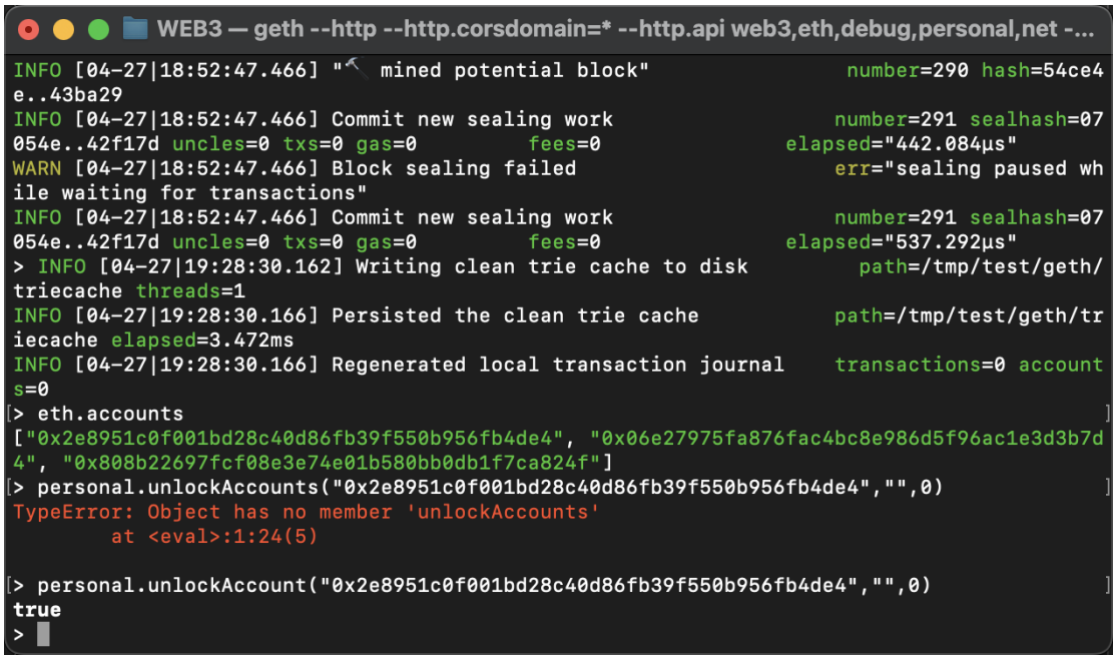
Demonstration is shown in the screenshot 30s video, to be more succinct.

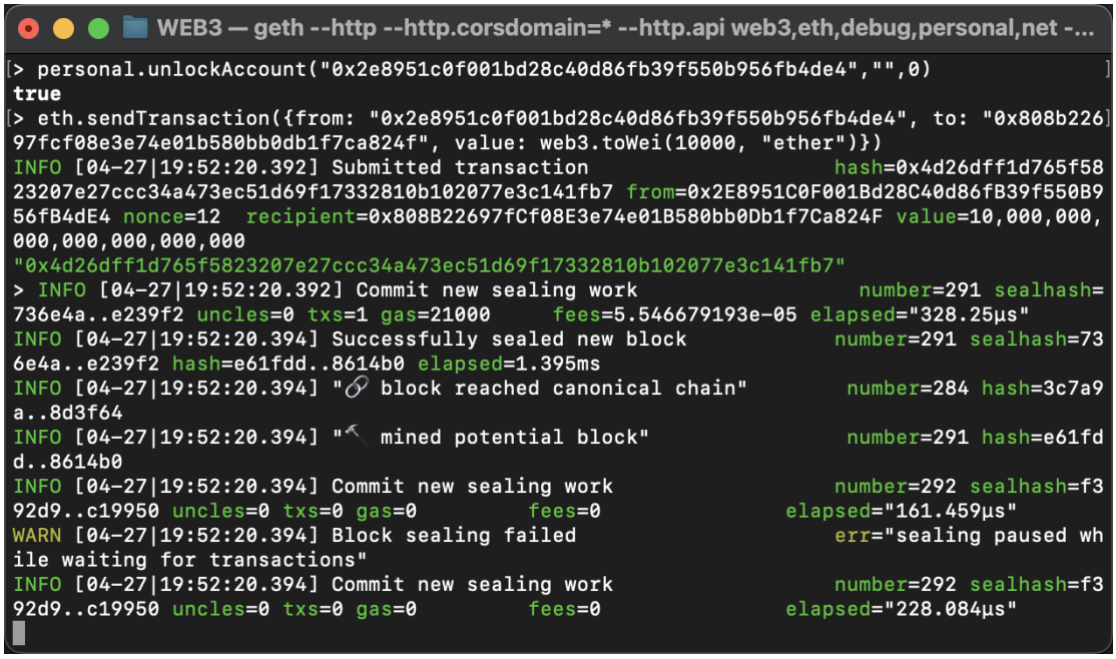
Step1: Geth start



Step1.1: Create accounts
Step1.2 Import and unlock accounts (the accounts in JSX file wallet)
Step1.3 Transfer ETH to the unlocked accounts to get initial funds

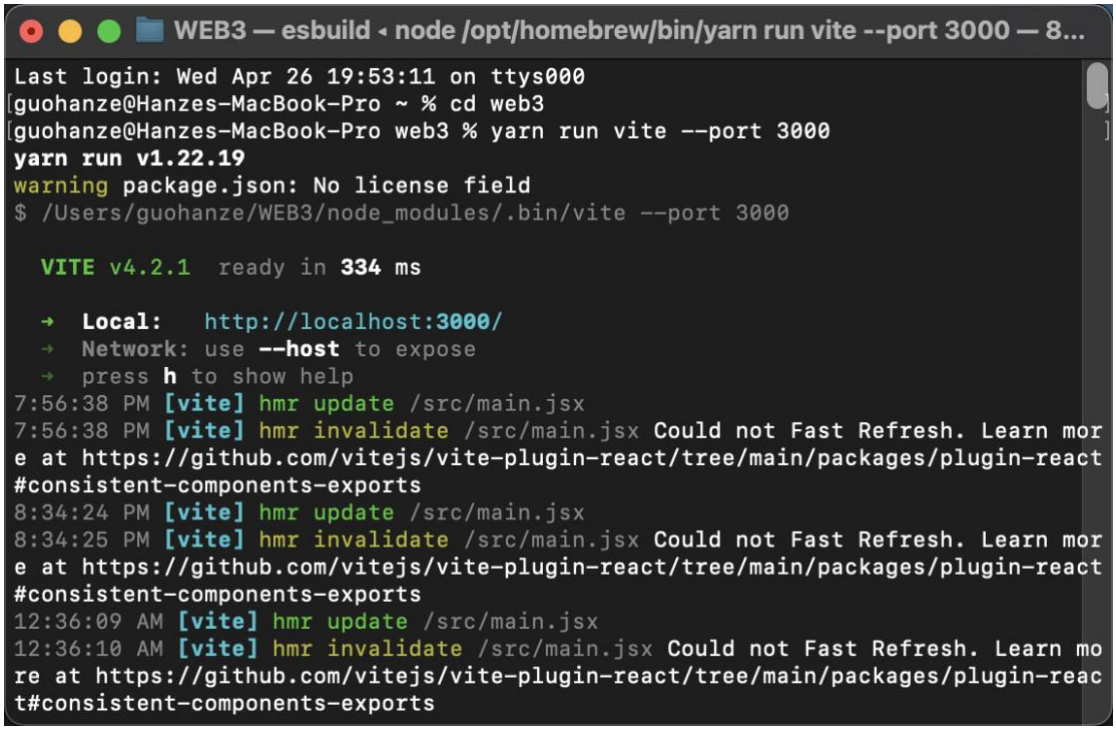






Total 3 accounts, the first one is the banker, others are the players.

Step2: Run Vite

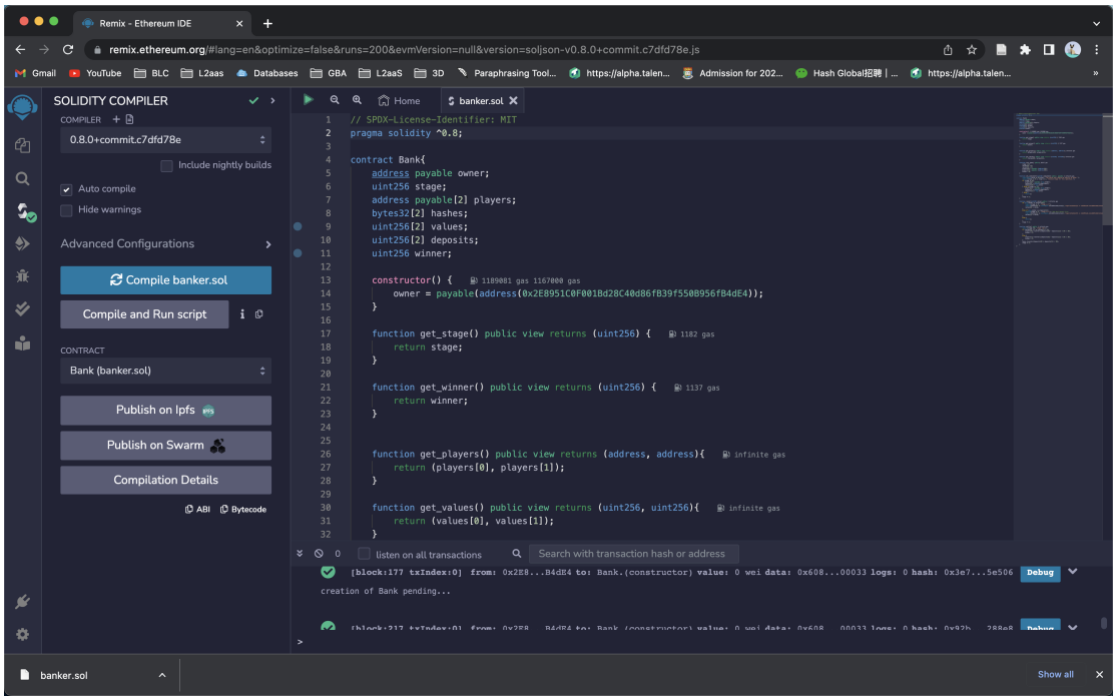


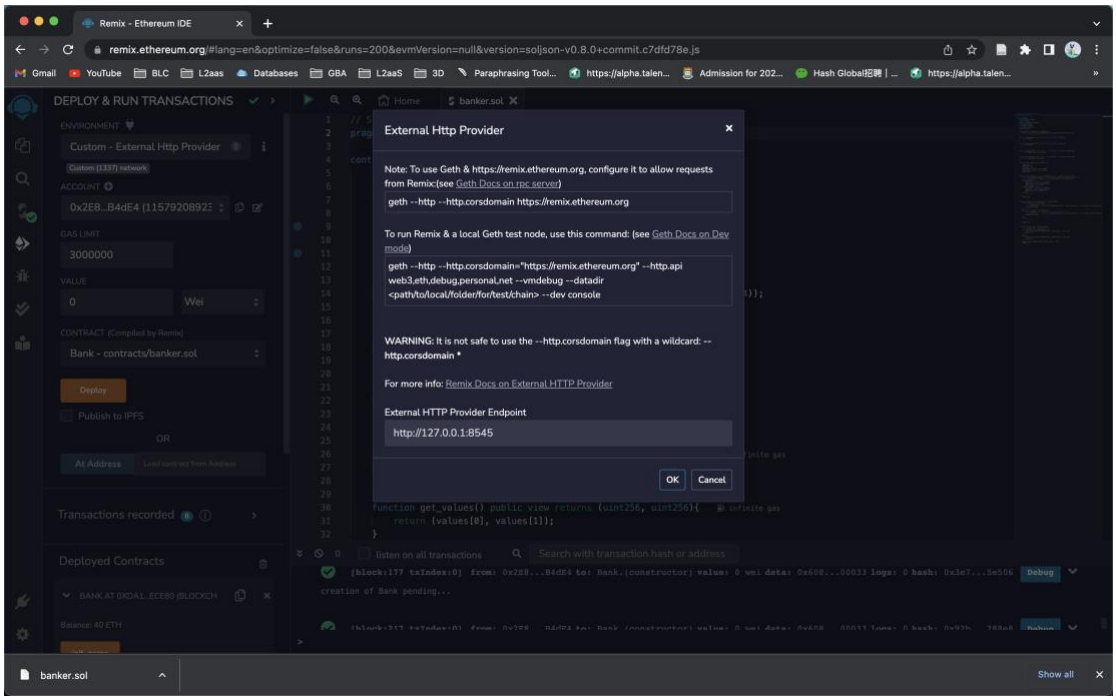
Step3: Run Remix

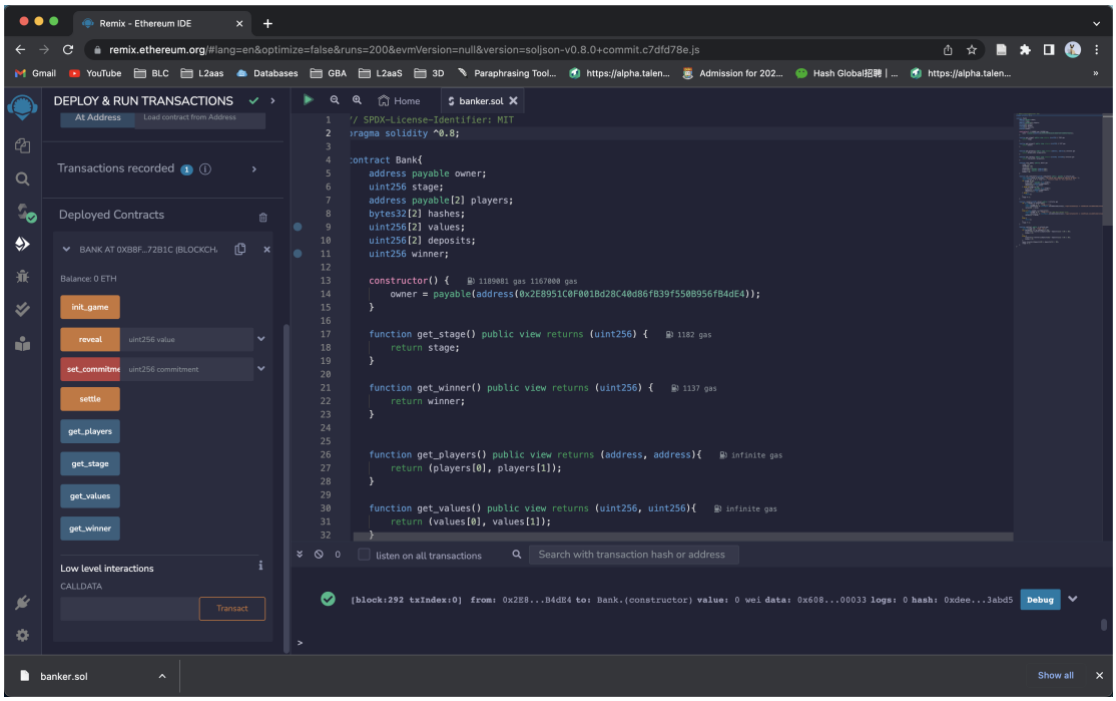
Step3.1: Compile the smart contract ‘banker.sol’

Step3.2: Deploy the contract on the provider

Step3.3: Set the JSX file parameters







Step4: Play Demo on ‘http://localhost:3000/’

Demonstration is shown in the screenshot 30s video, please have a check.

The question in Guan's PDF, Phase 2, Requirement 1, the following is the answer:

Question:
Please explain/prove whether this protocol actually works or not, in term of fairness, and what are the possible attacks (cheating) among the three parties. In your implementation, how have you solved them? You need to answer this question in your design document.

Answer:
The protocol is fair because it guarantees that neither party can cheat or manipulate the outcome of the game, assuming a reliable hash function is used. This is because the hash function is a one-way function, meaning that it is infeasible to determine the input value that corresponds to a given hash value. As a result, the parties cannot change their chosen number after sending the hash value to the banker without it being detected in Round 2 when the hash value is revealed along with the chosen number. Therefore, the winner is determined by the outcome of the coin flip based on the values chosen by the participants and the hash values sent to the banker.

However, there are still possible attacks that can occur in the protocol if it is not implemented correctly or if there are malicious actors involved. One possible attack is collusion between one of the parties and the banker, where they work together to cheat the other party. For example, the banker could provide one of the parties with the hash value of the other party's chosen number, allowing them to choose a number that will result in a win. This can be prevented by ensuring that the banker is trustworthy and does not collude with either party.
[In my implementation, the banker is the smart contract and it is impossible to reveal the number to anyone of the two players to collude, and the number is generated totally randomly.]

Another possible attack is the use of a weak hash function that can be easily reversed, allowing a party to determine the other party's chosen number from the hash value. This can be prevented by using a secure hash function that is resistant to preimage attacks, such as SHA-256.
[In my implementation, hash function is SHA-256.]

Finally, there is a risk that one of the parties may not reveal their chosen number in Round 2, which would prevent the outcome of the game from being determined. This can be addressed by having a time limit for revealing the chosen numbers and by having the banker notify all parties when the game is over.
[In my implementation, the smart contract will inform all two players when the game is over, and the state information will always be on the board.]