

Project Report BITS F463

Group Details - 37

Kartheek Sivavarma Nadimpalli 2018B4A70922H

Sai Nishanth Kurra 2018B4A20805H

Sai Rajesh 2018A4PS0296H

Problem Statement

Every year over 140 million people in India bet on various sports and this number goes as high as 370 million during major sporting events such as the Indian Premier League. With such a huge volume of betting being done, there are increased opportunities for fraud wherein betting companies may change balances or odds after the match to increase their profits or cheat certain individuals out of their winnings.

This project aims to combat this by creating a decentralized blockchain-based betting application that keeps user data secure and preserves data integrity.

Blockchain Introduction

Blockchain is a decentralized, distributed electronic database shared across a public or private network. Every transaction in a blockchain database is shared among a number of users, each one verifying that the database is accurate and preventing unauthorized transactions from being completed. The difference between a usual database and a blockchain is the way the data is structured. A blockchain stores data in the form of encrypted blocks that are chained together in a similar way to a linked list. A hash value is calculated based on all the data stored in a block. This hash is stored as one of the data items in the next block. Therefore, if any of the data values in the current block were to change, the hash value of the current block would change. This would in turn invalidate the previous hash value of the next block which would indicate that the chain has been tampered with. We cannot recompute the hash of the next block because we would need to do this for the entire chain and measures are put in place to make sure that this is too computationally intensive to be viable.

This property of blockchain allows data to be recorded and distributed, but not edited. Since the data is spread out across multiple nodes, if any person tried to tamper with the data, the point of error could be easily traced and fixed. Thus, blockchain creates an irreversible and immutable ledger of data. In order to make sure that blocks are not trivially added to the blockchain, we use a concept known as proof of work. This is basically a process that takes a sizeable amount of system resources and time to be completed. The process of creating a block by completing the proof of work process is known as mining. Successful mining is rewarded with some form of currency. This proof of work is what makes tampering with the blockchain so computationally intensive.

Applying Blockchain

In the context of our application, we wish to use blockchain to prevent malicious entities from tampering with the betting data. Each bet placed can be considered as a transaction. We store data such as user id, the team the user bets on, the bet amount, the odds at the time the bet is placed, etc. From the previous discussion, we know that the transactions stored in a blockchain cannot be edited. This ensures that the amount bet by a user and the odds at which the user bet cannot be changed. If the team the user bets on wins, the user can be assured that he/she will get the correct amount of money that they are owed. The proof of work is to find a value that will result in a specific prefix to the hash value of the block. A small amount of money is awarded for the successful completion of this. Before the transactions are added to the block, two types of verification are done, a normal type of verification to ensure that the details are all correct and an interactive zero-knowledge proof to check that the user knows what team they are betting for without revealing the actual details of the team.

Zero-Knowledge Proof

As a part of the verifying process, we want to make sure that the user knows which team they are betting on without the user actually revealing the team. This is achieved with the help of an interactive zero-knowledge proof. Each team that bets can be placed on has a team id. This team id is an integer. Let us call this integer 'x'. We want to verify that the user knows the value of x without actually revealing x. Two numbers p and g are selected. p is a prime number. For simplicity, the values of p and g have been hardcoded. We have chosen p to be 13 and g to be 31. A new variable y is calculated as $y = g^x \bmod p$. A random number between 2 and 7 is selected. Let this number be r. $c = g^r \bmod p$ and $c1 = g^{(r+x) \bmod (p-1)} \bmod p$ are calculated and sent. As part of the verification, we check that c1 matches with the value of $(c * y) \bmod p$. This process is repeated 10 times to ensure that the probability of getting it right by luck is too low. Thus we verify that the user has data about the team without revealing which team they have bet upon.

Code

The code consists primarily of three classes: the bet class, the block class and the blockchain class.

The bet class consists of a constructor and a calcHash() function.

The constructor initializes the bet data and has the following fields: user ID, game ID, bet amount, bet team, the odds of the bet in the form of numerator bet0 and a denominator bet1, the data and the status of the bet.

calcHash() uses the sha256 function to calculate a hash value based on the data initialized in the constructor function.

The block class consists of a constructor, calcHash() function and a mineBlock() function.

The constructor initializes the following fields: previous hash, timestamp, betTransactions, a dummy variable rands and hash of the current block.

calcHash() uses sha256 to calculate the hash of the current block based on the current data values of the block.

mineBlock() is the proof of work function. It keeps incrementing the value of the dummy variable rands till the hash value of the block starts with a required number of 'a's based on difficulty of the blockchain.

The Blockchain class has the following functions:

Constructor: initializes the chain with a genesis block, pending bets, the difficulty for the proof of work and the reward for mining.

createGenesisBlock(): Creates the first block of the blockchain with some arbitrary values.

getLatestBlock(): Retrieves the block that was most recently added to the blockchain.

minePendingBets(): This function mines a block by completing the proof of work and adding the block to the blockchain while also rewarding the user by depositing money in his account balance.

addBet(): Adds a bet placed to the list of pending bets. The user's account balance is verified to make sure the bet amount can be paid and the amount is deducted from their account.

getBalanceOfUser(): Retrieves the amount of money in the user's account.

completeGame(): We update the status of the game, declare the winner and update all user's balances based on whether they won or lost and the odds with which they placed their bets.

isChainValid(): Checks whether the blockchain is valid based on whether the previous hash is equal to the current block's previous hash data field and computing the current hash and checking whether it is equal to the value in the current hash variable.

A **user.json** file consists data pertaining to each user. It contains the name, password and **userID**. The **users.js** file gives each user an initial balance of 1000 rupees. Similarly, the **game.json** file consists of data pertaining to each game. It has two fields, the name of the game and a game ID field. The **games.js** file creates a team ID for each team in the game with a random number generator. This team ID is used later in the zero knowledge proof.

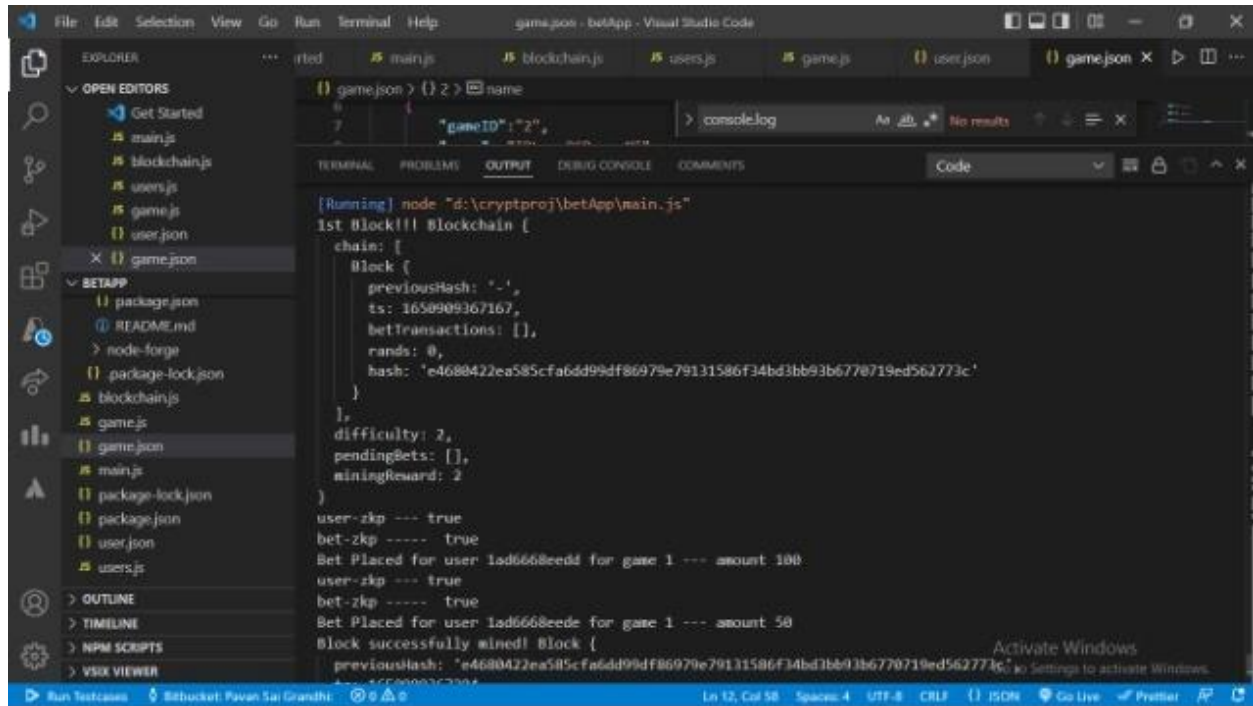
betOddsGenerator(): This function generates the odds for the game. Let's say there is a team A and team B, Let the odds of team A winning to team B winning is 1:5. Now the odds are calculated in this manner. Let's say a person bets 100 rs on team B with these odds. Then if team B wins he gets $(1 + \frac{1}{5}) * 100 = 120$ rs which means he has made a profit of 20 rs and odds is $1 + \frac{1}{5} = 1.2x$. Now if team B loses then he does not receive any money. Similarly if a person bets 100 rs on team A with the same odds then he gets $(1 + \frac{5}{1}) * 100$ which is 600 rs and the odds are 6x.

getUsersPL(userid): This function goes through every block in the blockchain and calculates the amount won or lost for each game that has been completed. It shows incomplete data for a pending game.

user_zkp(): This function does zero knowledge proof by using **userid** as x.

bet_zkp(): This function does zero knowledge proof to check if a bet is a valid bet by using the random number generated for each team as x. (which is the team's key)

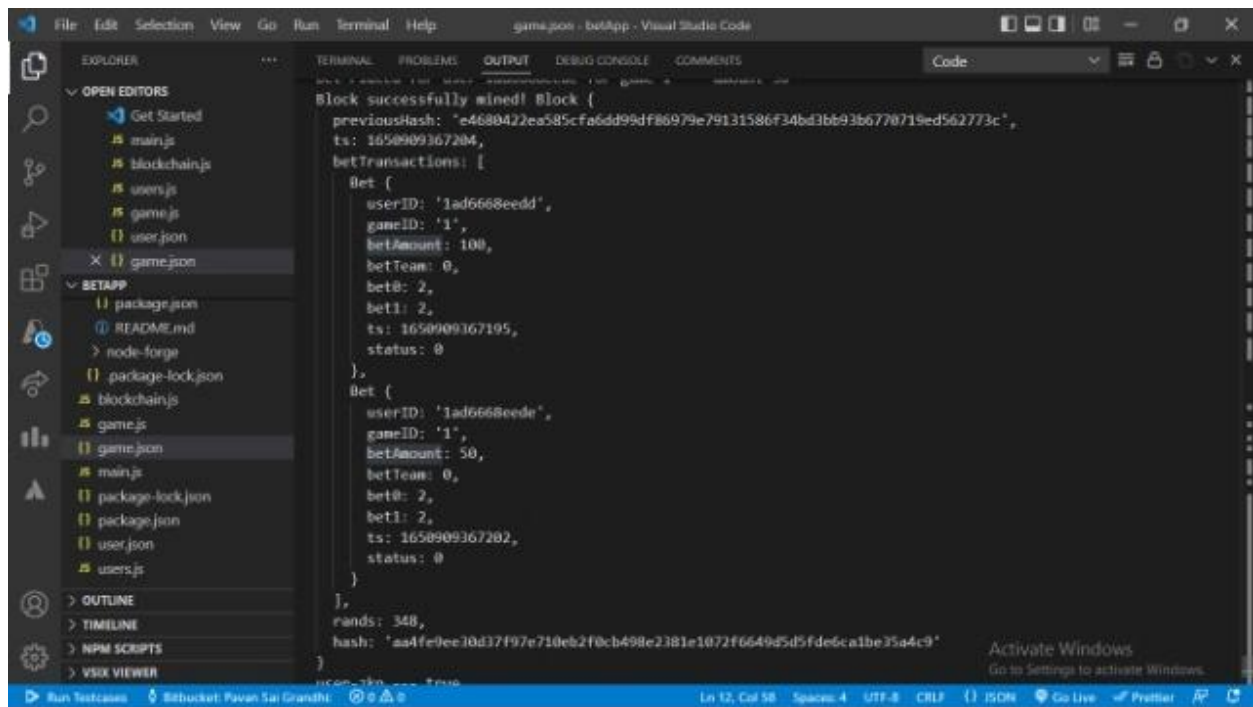
WORKING SCREENSHOTS:



This screenshot shows the Visual Studio Code interface with the 'game.json' file open. The file contains a JSON object with a 'gameID' of '2'. The terminal output shows the execution of 'main.js' and the creation of the first block in the blockchain.

```
game.json > {} > name
{
  "gameID": "2",
}
```

```
[Running] node "d:\cryptproj\betApp\main.js"
1st Block!!! Blockchain {
  chain: [
    Block {
      previousHash: '-',
      ts: 165809367167,
      betTransactions: [],
      rands: 0,
      hash: 'e4680422ea585cfa6dd99df86979e79131586f34bd3bb93b6770719ed562773c'
    }
  ],
  difficulty: 2,
  pendingBets: [],
  miningReward: 2
}
user-zkp --- true
bet-zkp ----- true
Bet Placed for user 1ad6668eedd for game 1 --- amount 100
user-zkp --- true
bet-zkp ----- true
Bet Placed for user 1ad6668eede for game 1 --- amount 50
Block successfully mined! Block {
  previousHash: 'e4680422ea585cfa6dd99df86979e79131586f34bd3bb93b6770719ed562773c',
  ts: 165809367204,
  betTransactions: [
    Bet {
      userID: '1ad6668eedd',
      gameID: '1',
      betAmount: 100,
      betTeam: 0,
      bet0: 2,
      bet1: 2,
      ts: 165809367195,
      status: 0
    },
    Bet {
      userID: '1ad6668eede',
      gameID: '1',
      betAmount: 50,
      betTeam: 0,
      bet0: 2,
      bet1: 2,
      ts: 165809367202,
      status: 0
    }
  ],
  rands: 348,
  hash: 'aa4fe0ee30d37f97e710eb2f0cb498e2381e1072f6649d5d5fde6ca1be35a4c9'
```



This screenshot shows the Visual Studio Code interface with the 'game.json' file open. The file now contains a JSON object with a 'gameID' of '1'. The terminal output shows the execution of 'main.js' and the creation of the second block in the blockchain.

```
game.json > {} > name
{
  "gameID": "1",
}
```

```
[Running] node "d:\cryptproj\betApp\main.js"
2nd Block!!! Blockchain {
  chain: [
    Block {
      previousHash: '-',
      ts: 165809367167,
      betTransactions: [],
      rands: 0,
      hash: 'e4680422ea585cfa6dd99df86979e79131586f34bd3bb93b6770719ed562773c'
    },
    Block {
      previousHash: 'e4680422ea585cfa6dd99df86979e79131586f34bd3bb93b6770719ed562773c',
      ts: 165809367204,
      betTransactions: [
        Bet {
          userID: '1ad6668eedd',
          gameID: '1',
          betAmount: 100,
          betTeam: 0,
          bet0: 2,
          bet1: 2,
          ts: 165809367195,
          status: 0
        },
        Bet {
          userID: '1ad6668eede',
          gameID: '1',
          betAmount: 50,
          betTeam: 0,
          bet0: 2,
          bet1: 2,
          ts: 165809367202,
          status: 0
        }
      ],
      rands: 348,
      hash: 'aa4fe0ee30d37f97e710eb2f0cb498e2381e1072f6649d5d5fde6ca1be35a4c9'
    }
  ],
  difficulty: 2,
  pendingBets: [],
  miningReward: 2
}
user-zkp --- true
bet-zkp ----- true
Bet Placed for user 1ad6668eedd for game 1 --- amount 100
user-zkp --- true
bet-zkp ----- true
Bet Placed for user 1ad6668eede for game 1 --- amount 50
Block successfully mined! Block {
  previousHash: 'e4680422ea585cfa6dd99df86979e79131586f34bd3bb93b6770719ed562773c',
  ts: 165809367204,
  betTransactions: [
    Bet {
      userID: '1ad6668eedd',
      gameID: '1',
      betAmount: 100,
      betTeam: 0,
      bet0: 2,
      bet1: 2,
      ts: 165809367195,
      status: 0
    },
    Bet {
      userID: '1ad6668eede',
      gameID: '1',
      betAmount: 50,
      betTeam: 0,
      bet0: 2,
      bet1: 2,
      ts: 165809367202,
      status: 0
    }
  ],
  rands: 348,
  hash: 'aa4fe0ee30d37f97e710eb2f0cb498e2381e1072f6649d5d5fde6ca1be35a4c9'
```

```

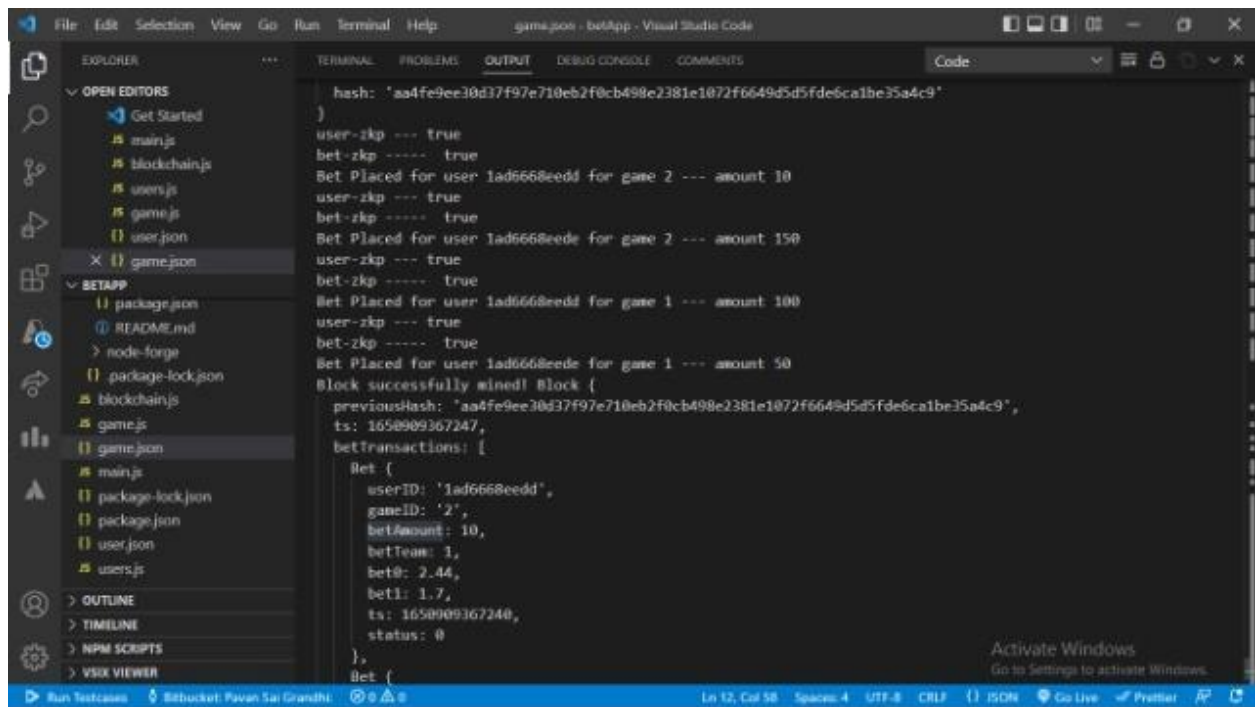
//
let {
  userID: '1ad6668eede',
  gameId: '1',
  betAmount: 50,
  betTeam: 0,
  bet0: 4.41,
  bet1: 1.29,
  ts: 1650909367246,
  status: 0
}
},
rands: 929,
hash: 'aad69ab0f8c3032f7612ac731806b68213ac1d9f5870b3beefdb65d9ccc1f49d'
}
Game 1 is over and team 1 has won and user balances updated.
Is Chain valid --- true
User's P&L statement for user -- 1ad6668eedd [
  { gameId: 'IPL - CSK vs MI', betAmount: 100, wl: 'lost', pl: -100 },
  { gameId: 'IPL - RCB vs MI', betAmount: 10, wl: '-', pl: '-' },
  { gameId: 'IPL - CSK vs MI', betAmount: 100, wl: 'Won', pl: 29 }
]
User's P&L statement for user -- 1ad6668eede [
  { gameId: 'IPL - CSK vs MI', betAmount: 50, wl: 'lost', pl: -50 },
  { gameId: 'IPL - RCB vs MI', betAmount: 150, wl: '-', pl: '-' },
  { gameId: 'IPL - CSK vs MI', betAmount: 50, wl: 'lost', pl: -50 }
]
User's P&L statement for user -- 1ad6668eedf []

```

```

},
status: 0
},
let {
  userID: '1ad6668eede',
  gameId: '2',
  betAmount: 150,
  betTeam: 0,
  bet0: 2.44,
  bet1: 1.7,
  ts: 1650909367243,
  status: 0
},
let {
  userID: '1ad6668eedd',
  gameId: '1',
  betAmount: 100,
  betTeam: 1,
  bet0: 4.41,
  bet1: 1.29,
  ts: 1650909367245,
  status: 0
},
let {
  userID: '1ad6668eede',
  gameId: '1',
  betAmount: 50,
  betTeam: 0,
  bet0: 4.41,
  bet1: 1.29,

```



```
hash: 'aa4fe9ee30d37f97e710eb2f0cb498e2381e1072f6649d5d5fde6ca1be35a4c9'
}
user-zkp --- true
bet-zkp ----- true
Bet Placed for user 1ad6668eedd for game 2 --- amount 10
user-zkp --- true
bet-zkp ----- true
Bet Placed for user 1ad6668eedd for game 2 --- amount 150
user-zkp --- true
bet-zkp ----- true
Bet Placed for user 1ad6668eedd for game 1 --- amount 100
user-zkp --- true
bet-zkp ----- true
Bet Placed for user 1ad6668eedd for game 1 --- amount 50
Block successfully mined! Block {
  previousHash: 'aa4fe9ee30d37f97e710eb2f0cb498e2381e1072f6649d5d5fde6ca1be35a4c9',
  ts: 1650909367247,
  betTransactions: [
    Bet {
      userID: '1ad6668eedd',
      gameID: '2',
      betAmount: 10,
      betTeam: 1,
      bet0: 2.44,
      bet1: 1.7,
      ts: 1650909367240,
      status: 0
    },
    Bet {
```

Flow Diagram

